Modelos para Sistemas Comunicantes

Paulo Maciel

prmm@cin.ufpe.br

www.modcs.org

Centro de Informática

Universidade Federal de Pernambuco

Objetivo

- Apresentar formalismos para modelar e avaliar sistemas concorrentes.
- Descrever as principais características destes modelos.
- Modelar e avaliar problemas.
- Destacar as pontencialidades e restrições destes modelos.

Conteúdo

- Conceitos sobre Sistemas e Modelos
- Autômatos concorrentes
- Introdução as álgebras de processos
- Redes de Petri

Conteúdo

- Redes de Petri
  - Classes das redes de Petri
  - Conceitos básicos
  - Propriedades estruturais e comportamentais
  - Métodos de análise
  - *Coloured Petri nets*
  - Redes de Petri temporizadas

Metodologia

- Aulas expositivas
- Aulas práticas.

Avaliação

- Listas
- Trabalho Final (*draft* de artigo)

Bibliografia Básica

- Intoduction to Discrete Event Systems, Cassandras and Lafortune, Kluwer, 2008.
- Concurrency: State Models & Java Programs, 2nd Edition byJeff MageeandJeff Kramer John Wiley & Sons 2006
- Uma Introdução às Redes de Petri. Paulo Maciel, Rafael Lins, Paulo Cunha, Escola de Computação, 1996.
- Lectures Notes on Petri Nets I, Basic Models. Springer Verlag, 1998.
- Lectures Notes on Petri Nets II, Applications. Springer Verlag, 1998.

Classificação dos Modelos

- O que é um sistema?
- O que é um modelo?

Classificação dos Modelos

- Estáticos
- Dinâmicos
  - Variantes no Tempo
  - Invariantes
    - Estado Contínuo
    - Estado Discreto
      - Tempo Discreto
      - Tempo Contínuo
        - Determinísticos
        - Probabilístico

Classificação dos Modelos

Estáticos Dinâmicos

Não-Temporais Invariantes no Tempo Variantes no Tempo

Eventos Discretos Contínuos

Estado Contínuo Estado Discreto

Tempo Discreto Tempo Contínuo

Determinísticos Probabilístico

Classificação dos Modelos

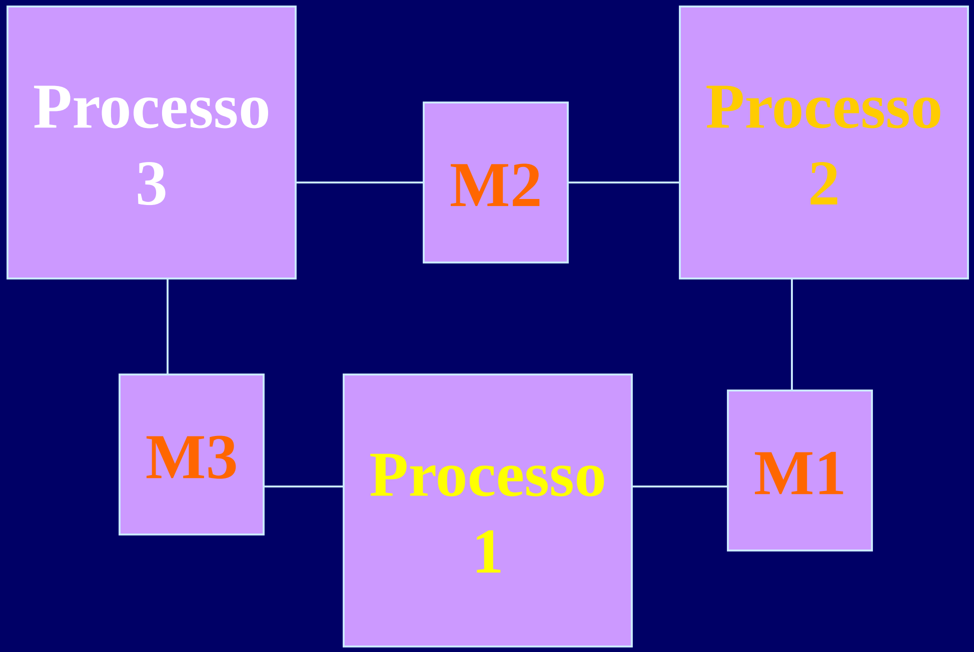
- Modelos Baseados em Estado
  - Consideram apenas os estados para modelar e se referir as propriedades do sistema.
  - Maioria das lógicas temporais, autômatos
- Modelos Baseados em Ações
  - Consideram apenas as ações para modelar e se referir as propriedades dos sistemas.
  - As álgebras de processos: CCS, CSP, COSY, FSP
- Modelos Heterogêneos
  - Consideram ações e estados.
  - Redes de Petri

Motivação

- Considere uma situação onde se deseja representar de forma precisa o comportamento de um sistema de manufatura, responsável pela fabricação de três tipos de produtos diferentes.
- A realização das atividades de manufatura de cada produto é denominada um processo. Estes processos podem ser executado paralelamente.
- O ambiente de manufatura disponibiliza três máquinas (recursos) para realização das atividades dos processos.
- Cada par de processos compartilha entre si uma máquina.
- E cada processo precisa simultaneamente de duas máquinas para realização de uma determinada atividade.

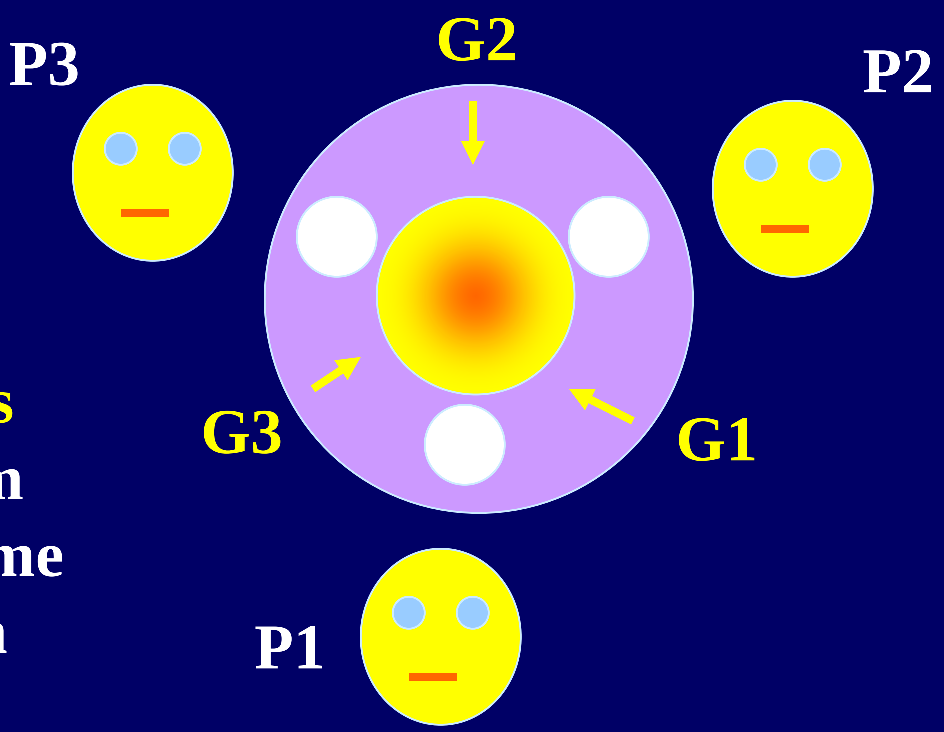
Motivação

- Estrutura do Problema



Motivação

- O Problema Jantar dos Filósofos



Filósofos

- **Pensam**
- **Tem fome**
- **Comem**

Como especificar adequadamente este problema de forma que o modelo obtido não trave (*deadlock*) e que todos os filósofos tenham oportunidade de comer ?

Apresentacão

- **LTS – Sistema de Transição Rotulado, Máquinas ou Autômatos** ***(Labeled Transitions Systems)***
- **FSP –** ***Finite Sequential Process***
- **Redes de Petri**

Máquinas de Estados, Autômatos ou Sistemas de Transição Rotulados

Máquinas de Estados

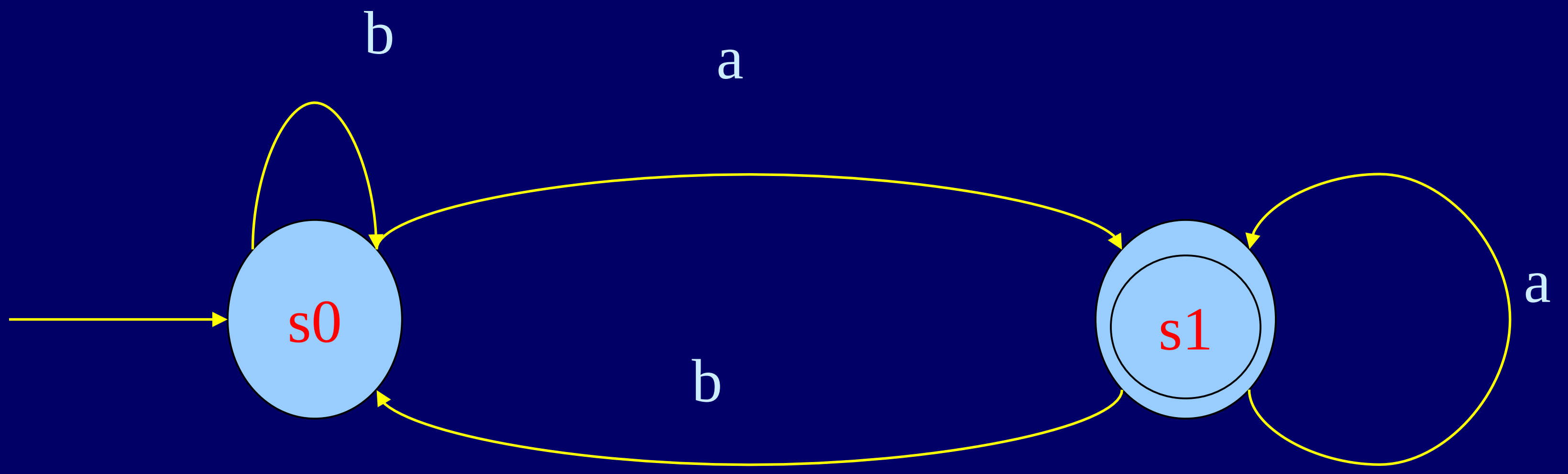
- Máquina de Estados Determinística
- Máquina de Estados Não-Determinística
- Máquina de Estados Finitos Não-Determinísitca
- Máquinas de Estados Finitos Determinística
  - Máquina de Estados Finitos Determinística com Entradas e Saídas

Máquina de Estados Determinística

- $SM = (S, E, f, \Gamma, s_0, S_m)$
  - S – Conjunto de estados
  - $s_0 \in S$ – Estado inicial
  - E – Alfabeto (conjunto de eventos)
  - $f : S \times E \rightarrow S$ – Função de próximo estado
  - $\Gamma : S \rightarrow 2^E$ – Função dos eventos factíveis
  - $S_m \subseteq S$ - Conjunto de estados marcados

Máquina de Estados Determinística

$S=\{s0,s1\}$, $E=\{a,b\}$, $f(s0,a) = s1$, $f(s0,b) = s0$, $f(s1,a)=s1$, $f(s1,b)=s0$, $\Gamma(s0) = \{a,b\}$, $\Gamma(s1) = \{a,b\}$, $S_m=\{s1\}$



Máquina de Estados Determinística

E^* denota o conjunto de todas as "strings" (sequências) finitas de elementos de E, incluindo a "strings" vazia ε.

E^* é denominado *Kleene-closure*.

Suponha $E = \{a, b, c\}$, portanto

$E^* = \{\, \varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \ldots \}$.

Por conveniência, estendemos f do domínio $X \times E$ para $X \times E^*$, da seguinte maneira:

$$f(x, \varepsilon) = x$$

$$f(x, se) = f(f(x, s), e)$$

para $s \in E^*$ e $e \in E$

Máquina de Estados Determinística

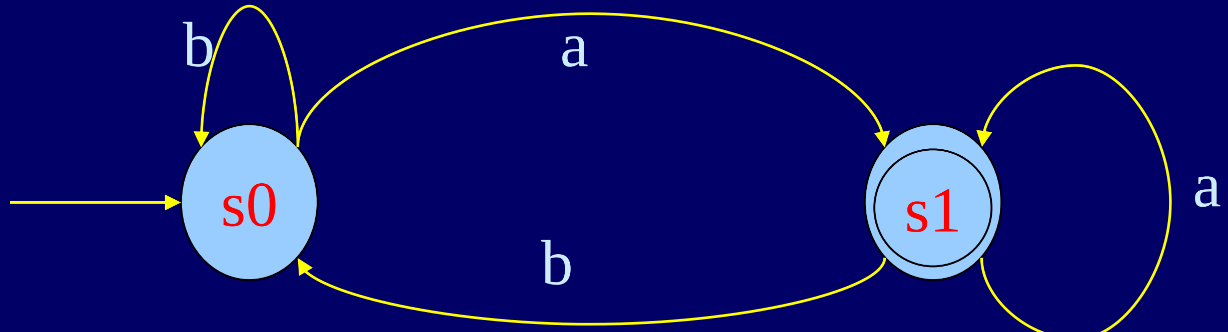
- Linguagem Gerada

$$L(SM) = \{st \in E^* \mid f(s_0, st) \text{ é definida}\}$$

se f for uma função total então $L(SM) = E^*$

- Linguagem Marcada

$$Lm(SM) = \{st \in L(SM) \mid f(s_0, st) \in S_m\}$$



$Lm(SM) = \{a, aa, ba, aaa, aba, ...\}$

$L(SM)= E^*$

Máquina de Estados Determinística

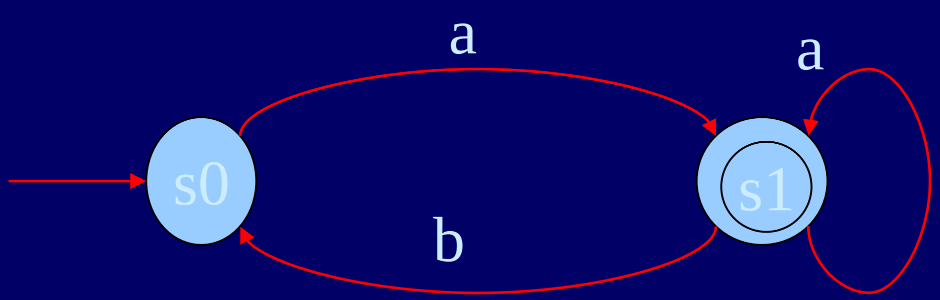
- Duas Máquinas de Estados SM1 e SM2 são ditas equivalentes se

$$L(SM1) = L(SM2)$$

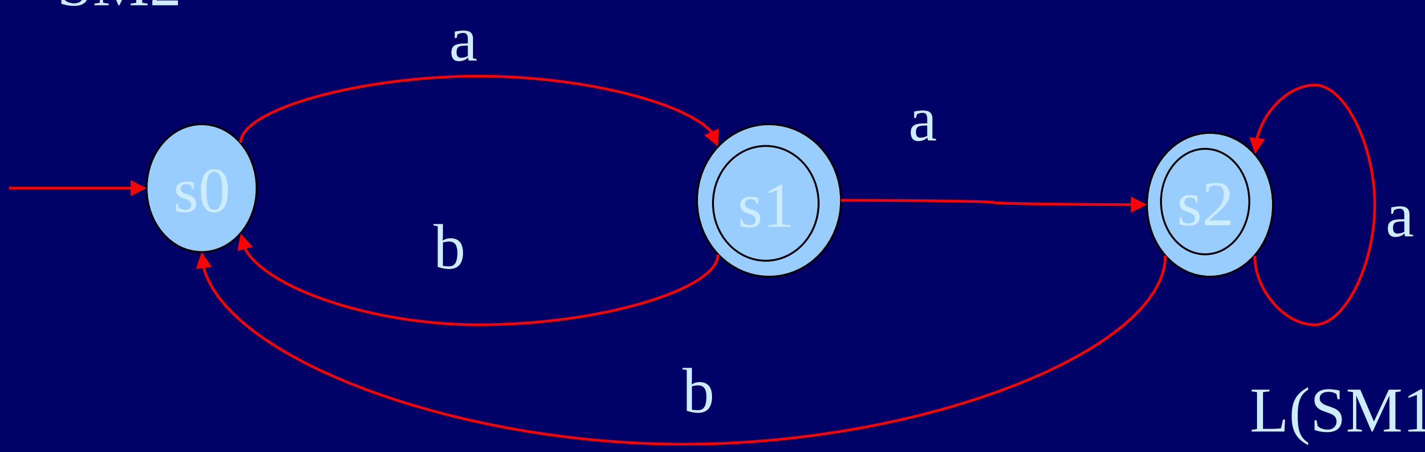
$$Lm(SM1) = Lm(SM2)$$

Máquina de Estados Determinística

SM1



SM2

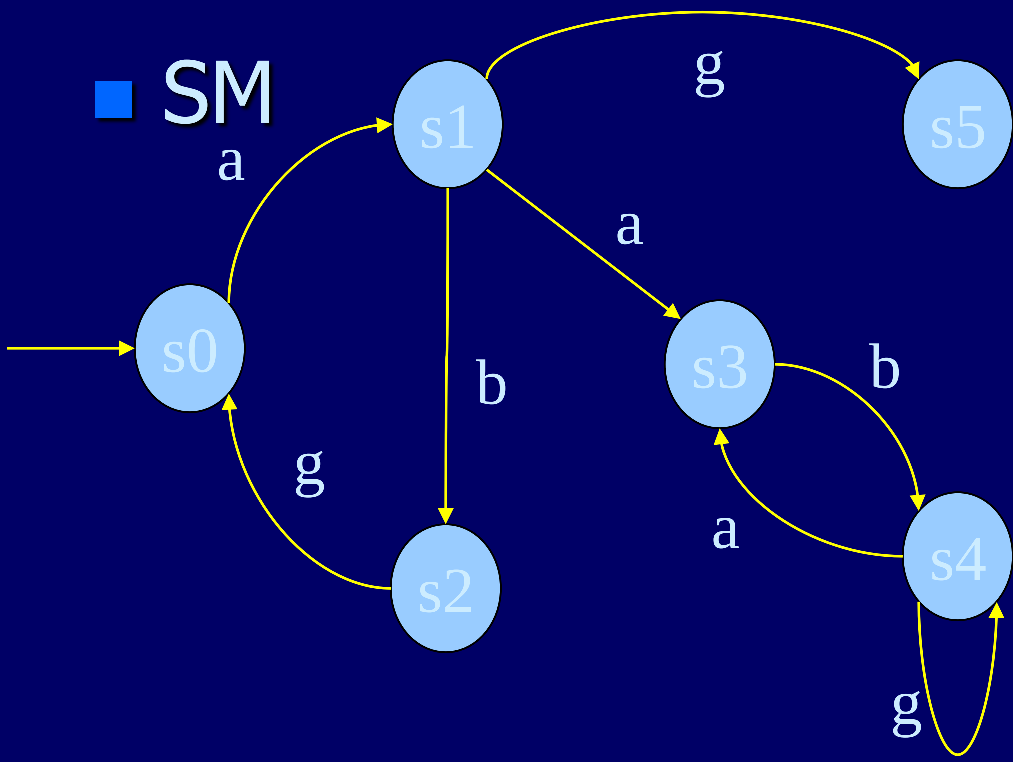


L(SM1) = L(SM2)

Lm(SM1)=Lm(SM2)

Máquina de Estados Determinística

- SM



•Deadlock - Estado s5

•Livelock - Estados s3, s4

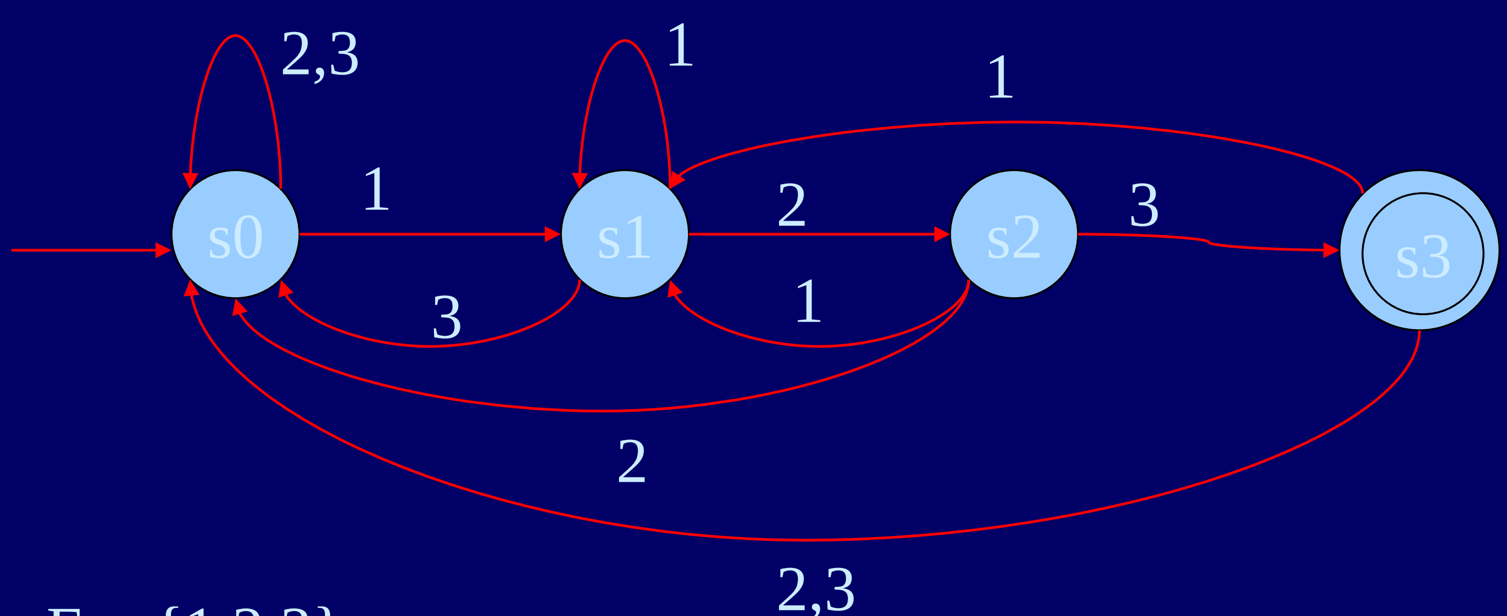
•Alcançabilidade - Um estado sn é alcançável de s0 se existe um conjunto de eventos que realizados a partir de s0 levam a sn.

$s4 \in R(s0)$

$s0 \notin R(s4)$

Máquina de Estados Determinística

- Detector de Seqüência 123



E = {1,2,3}

Máquina de Estados Determinística

Sistema de Fila

Neste ambiente, os usuários chegam e solicitam acesso a um servidor. Caso o servidor esteja ocupado, os usuários devem aguardar na fila.Quando o usuário completa a operação solicitada, ele sai do sistema e o próximo usuário da fila é servido imediatamente.

Podemos modelar este problema com um automata (máquina) de estados infinitos.

Os eventos que dirigem o sistema são:

a : chegada de um usuário

d: saída de um usuário

Portanto,

$E=\{a,d\}$

$S=\{0,1,2,...\}$

$f(s,a) = s+1, \forall\ s\geq 0$

$f(s,d) = s-1$, se $s>0$

$\Gamma(s) = \{a,d\}, \forall\ s>0$

$\Gamma(0) = \{a\}$

Máquina de Estados Determinística

Sistema de Fila

$E=\{a,d\}$

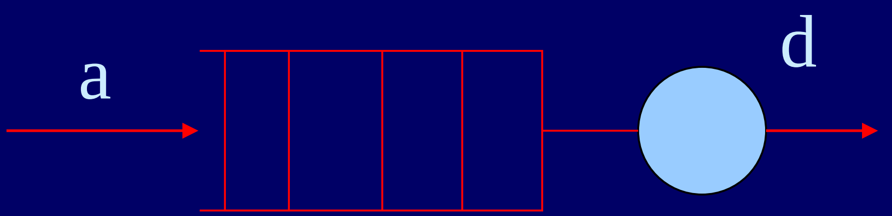
$S=\{0,1,2,\ldots\}$

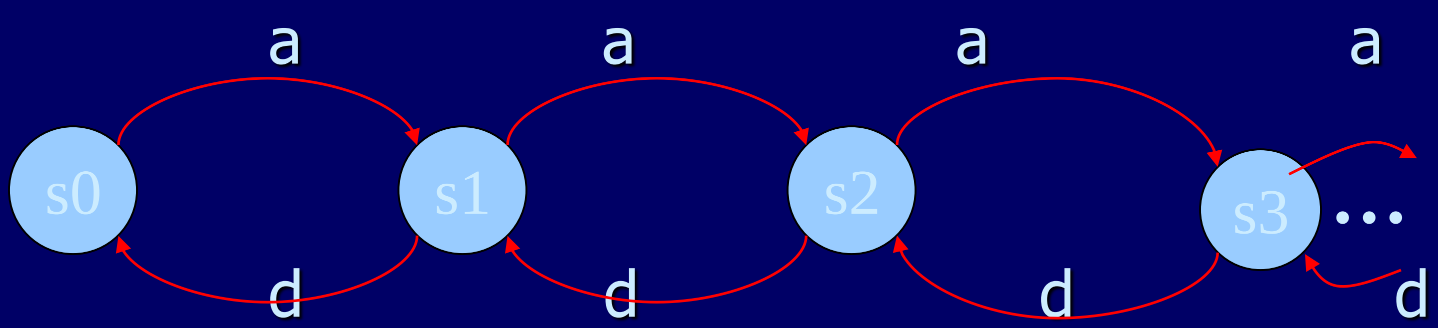
$f(s,a) = s+1, \ \forall \ s \geq 0$

$f(s,d) = s-1, \text{ se } s>0$

$\Gamma(s) = \{a,d\}, \ \forall \ s>0$

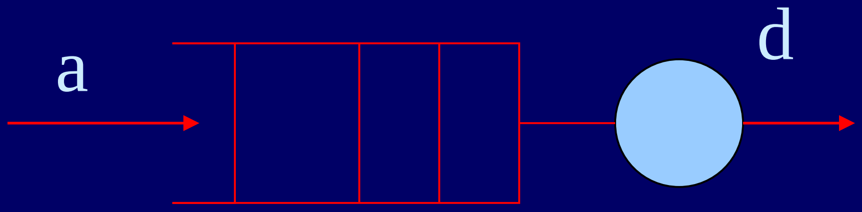
$\Gamma(0) = \{a\}$





Máquina de Estados Determinística

Sistema de Fila



Para este mesmo sistema, vamos focar o estado do servidor. Vamos assumir que o servidor pode estar desocupado (I), ocupado (B) ou quebrado (D). Assumi-se que quando o servidor está quebrado, o usuário em serviço é perdido. Portanto, após o reparo, o servidor está desocupado. Os eventos são: serviço começa (a), serviço finaliza (b), servidor quebra (l) e servidor é reparado (n).

$E=\{a,b,l,n\}$

$S=\{I,B,D\}$

$f(I,a) = B$, $f(B,b) = I$,

$f(D,n) = I$, $f(B,l) = D$,

$\Gamma(I) = \{a\}$, $\Gamma(B) = \{b,l\}$,

$\Gamma(D) = \{n\}$

Máquina de Estados Determinística

Sistema de Fila

E={a,b,l,n}

S={I,B,D}

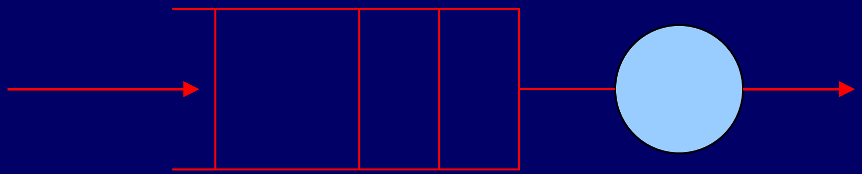
f(I,a) = B, f(B,b) = I,

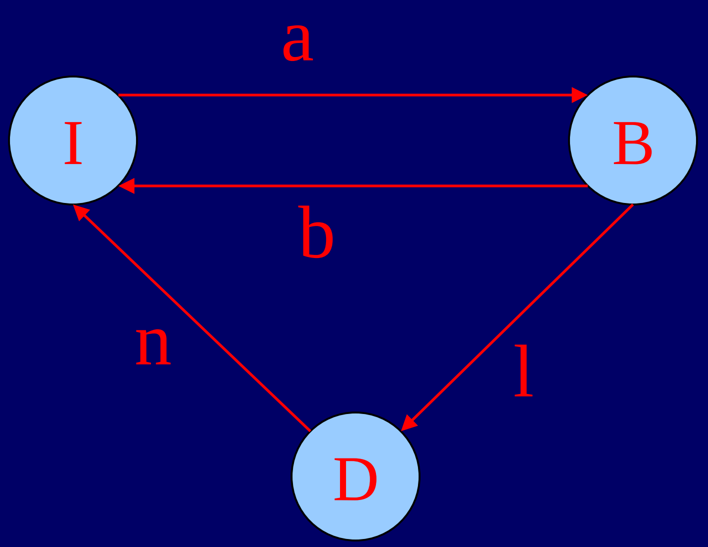
f(D,n) = I, f(B,l) = D,

Γ(I) = {a}, Γ(B) = {b,l},

Γ(D) = {n}

.





Máquina de Estados Não-Determinística

- $SM_{nd} = (S, E \cup \{\varepsilon\}, F, \Gamma, s_0, S_m)$
  - S – Conjunto de estados
  - $s_0 \in S$ – Estado inicial
  - E – Alfabeto (conjunto de eventos)
  - $F : S \times E \times S$ – Relação de próximos estados
  - $\Gamma : S \rightarrow 2^E$ - Função dos eventos factíveis
  - $S_m \subseteq S$ - Conjunto de estados marcados

Por quê utilizar?

1. Desconhecimento sobre determinadas atividades ou necessidade de abstração
2. É possível obter um automata de menor número de estados.

Máquina de Estados Não-Determinística

- $SM_{nd} = (S, E \cup \{\varepsilon\}, F, \Gamma, s_0, S_m)$

$S=\{0,1\}$

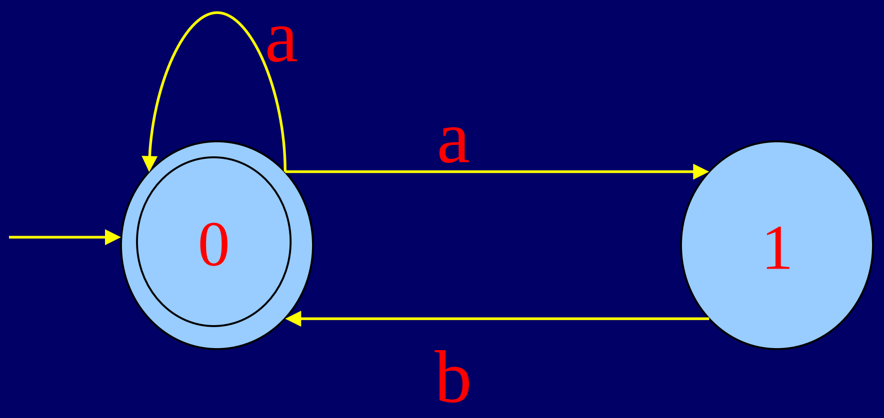
$E=\{a,b\}$

$F(0,a) = \{0,1\}$

$F(1,b) = \{0\}$

$\Gamma(0) = \{a\}$

$\Gamma(1) = \{b\}$



Máquina de Estados Não-Determinística

- Máquina determinísitca equivalente

$SM_{nd}= (S,E,f,\Gamma, s_0,S_m)$

$S=\{A,B\}$

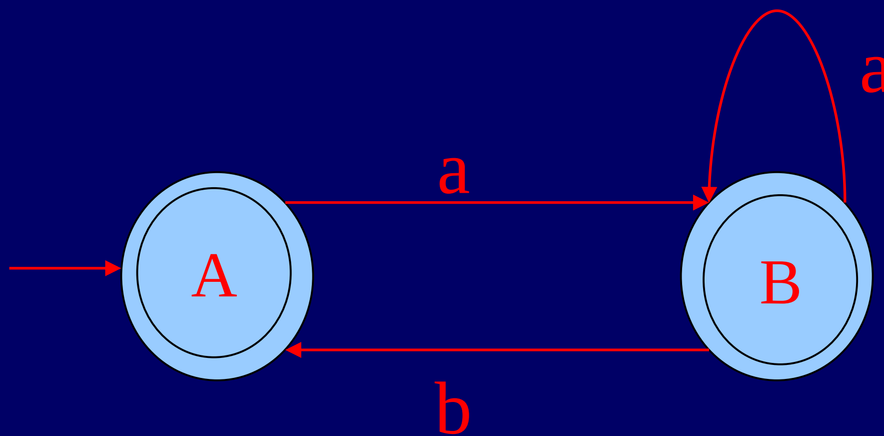
$E=\{a,b\}$

$f(A,a) = B$

$f(B,a) = A$, $f(B,b) = A$

$\Gamma(A) = \{a\}$

$\Gamma(B) = \{a,b\}$



Máquinas não-determinísticas podem ser convertidas em máquinas deteminísticas.

Máquina de Estados
-Algumas Operações-

- Partes Acessíveis (*Reachable*)

$SM = (S, E, f, s_0, S_m)$

$Rc(SM) = (S_{rc}, E, f_{rc}, s_0, S_m^{rc})$

$S_{rc} = \{s \in S \mid \exists\ st \in E^*$ que $f(s_0, st)$ esteja definida$\}$

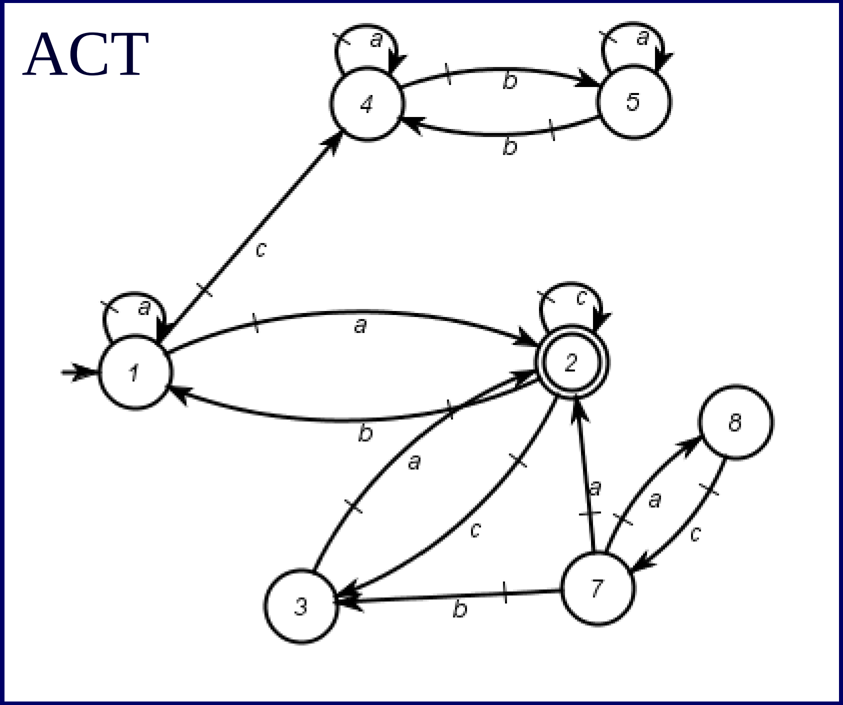
$S_m^{rc} = S_m \cap S_{rc}$

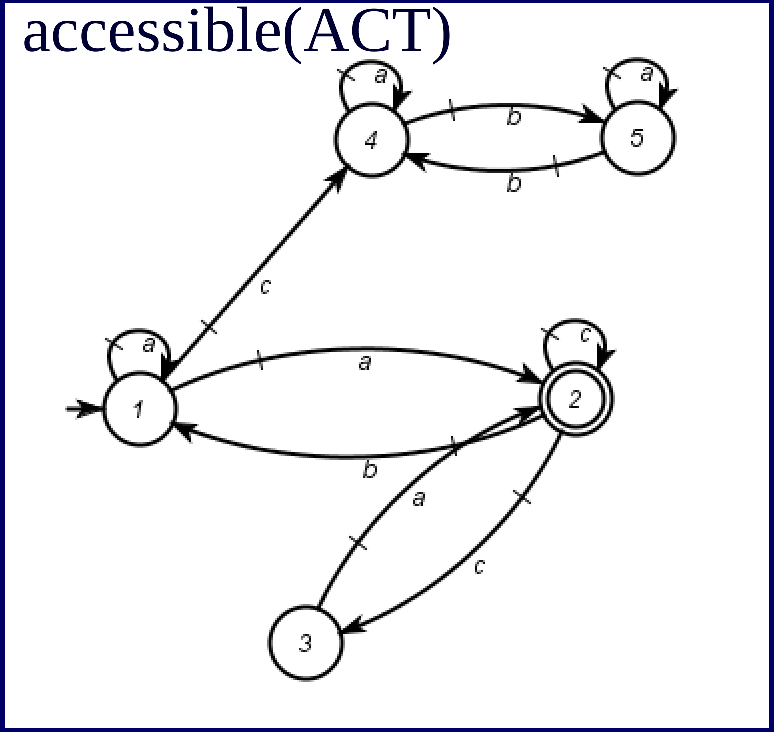
$f_{rc} = f|_{S_{rc} \times E \to S_{rc}}$ é a função f com o domínio restrito a (S_{rc}, E)

Nós vamos assumir que as máquinas tratadas aqui são acessíveis.

Máquina de Estados
-Algumas Operações-

- Partes Acessíveis (*Reachable*)





Máquina de Estados
-Algumas Operações-

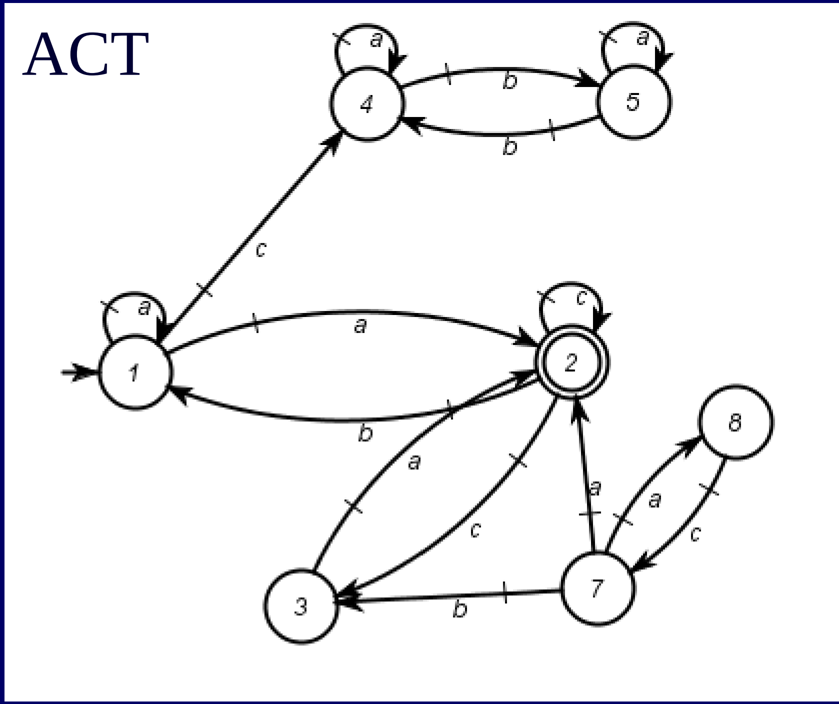
- Partes Co-Acessíveis (Co-*Reachable*)

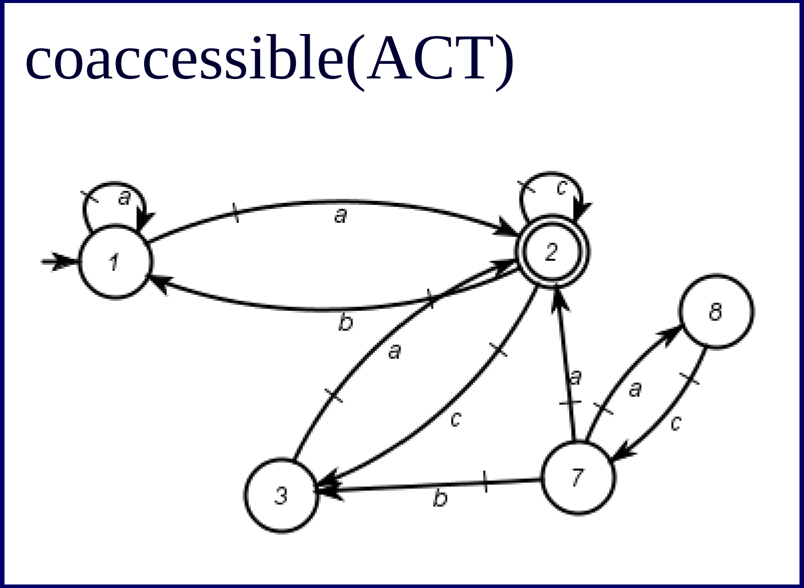
É um autômata em que todos os estados marcados são alcançáveis, ou seja:

$f_{rc}(s_{0,}s)=s_m$ está definida para $\forall st \in E^*$, onde $S_m \in S_m$

Máquina de Estados
-Algumas Operações-

- Partes Co-Acessíveis (Co-*Reachable*)

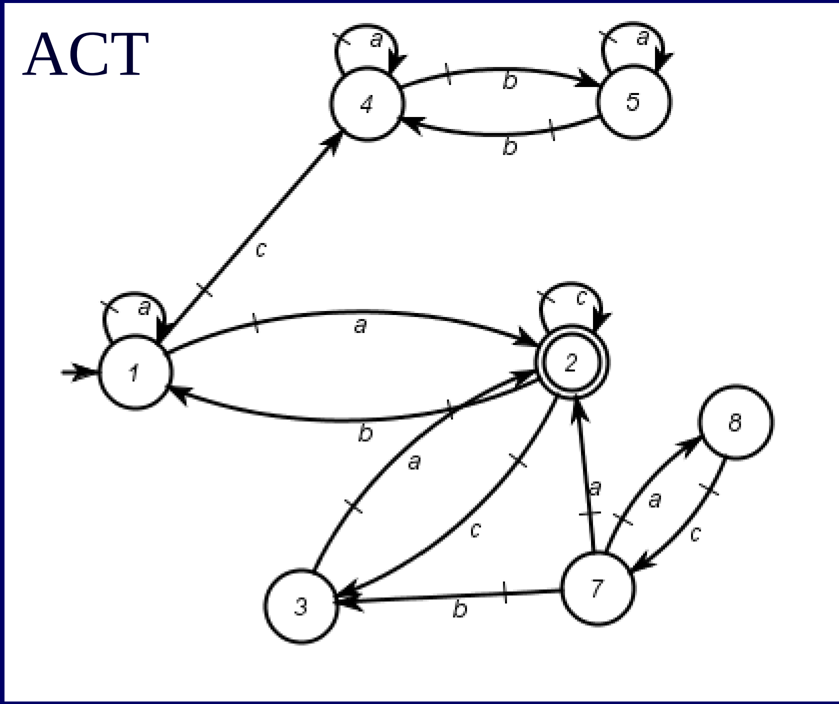


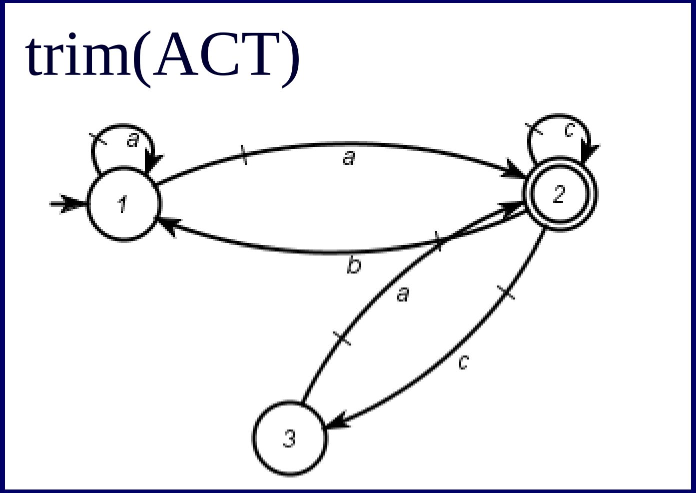


Máquina de Estados

-Algumas Operações-

- Operador Trim remove todos os estados que não são alcançáveis ou co-alcançáveis.





Máquina de Estados
-Algumas Operações-

- Composição Paralela

$SM_1 = (S_1, E_1, f_1, \Gamma_1, s_0^1, S_m^1)$

$SM_2 = (S_2, E_2, f_2, \Gamma_2, s_0^2, S_m^2)$

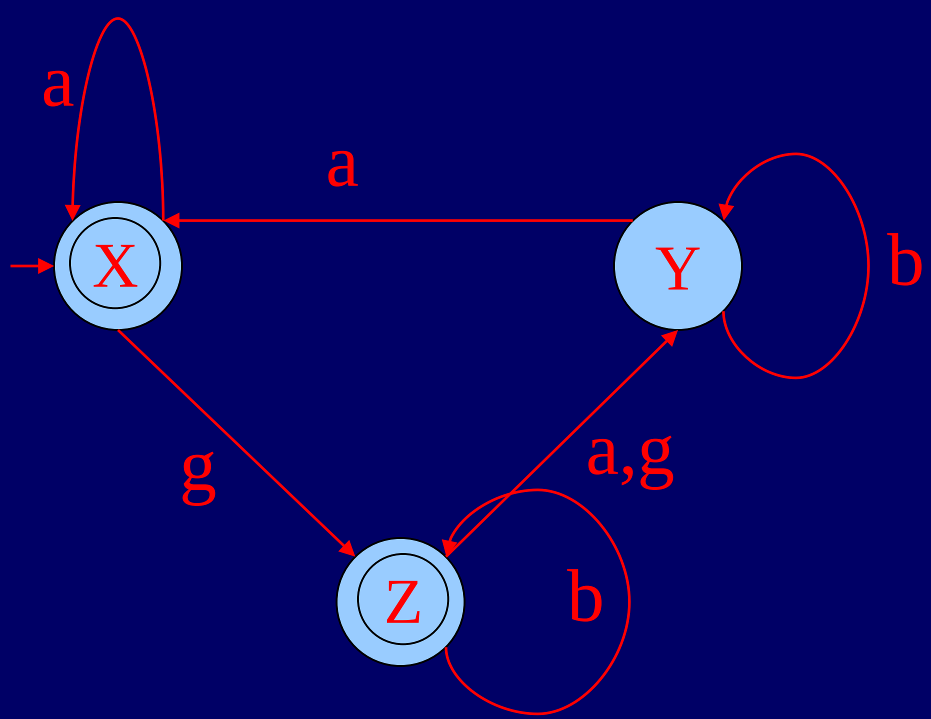
$SM_1 || SM_2 = Rc(S_1 \times S_2, E_1 \cup E_2, f_{1||2}, \Gamma_{1||2}, (s_0^1, s_0^2), S_m^1 \times S_m^2)$

$$f_{1||2}((s_1, s_2), e) = \begin{cases} (f_1(s_1, e), f_2(s_2, e)) & \text{se } e \in \Gamma_1(s_1) \cap \Gamma_2(s_2) \\ (f_1(s_1, e), s_2)) & \text{se } e \in \Gamma_1(s_1) \setminus E_2 \\ (s_1, f_2(s_2, e)) & \text{se } e \in \Gamma_2(s_2) \setminus E_1 \\ \text{indefinido} & \text{caso contrário} \end{cases}$$

$\Gamma_{1||2}(s_1, s_2) = \{\Gamma_1(s_1) \cap \Gamma_2(s_2)\} \cup \{\Gamma_1(s_1) \setminus E_2\} \cup \{\Gamma_2(s_2) \setminus E_1\}$

Máquina de Estados
-Algumas Operações-

- Composição Paralela



$S=\{X,Y,Z\}$, $E=\{a,b,g\}$,
$f(X,a) = X$, $f(X,g) = Z$,
$f(Y,a)=X$, $f(Y,b)=Y$,
$f(Z,b)=Z$,
$f(Z,a)=f(Z,g)=Y$

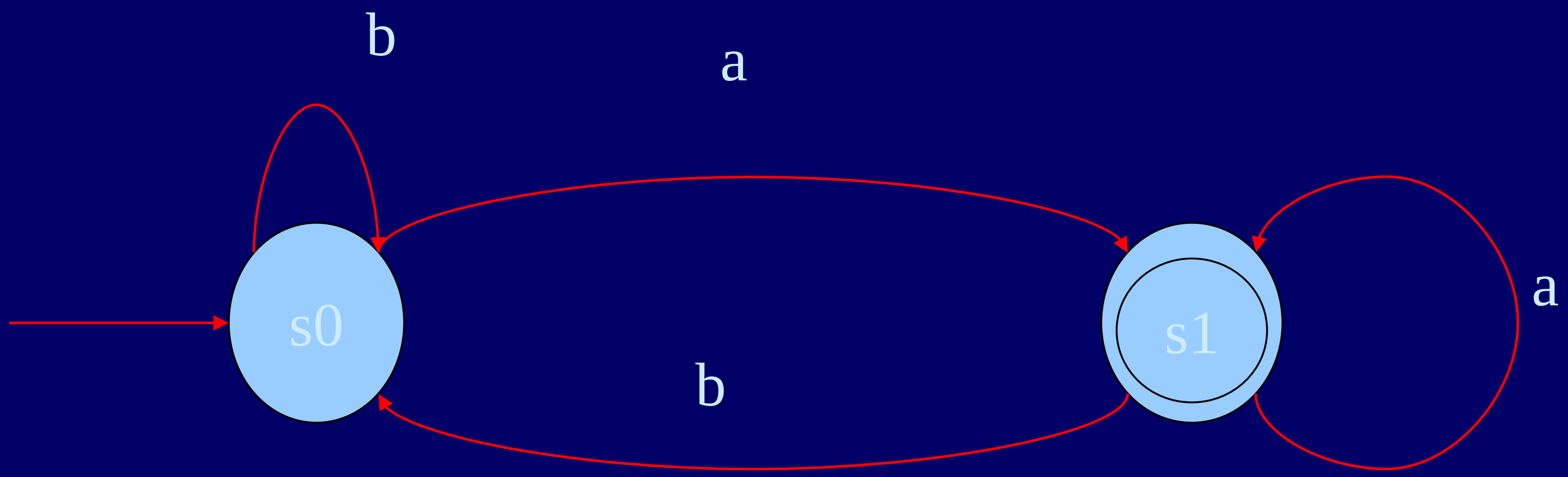
$\Gamma(X) = \{a,g\}$, $\Gamma(Y) = \{a,b\}$,
$\Gamma(Z) = \{a,b,g\}$

$S_m=\{X,Z\}$

Máquina de Estados
-Algumas Operações-

- Composição Paralela

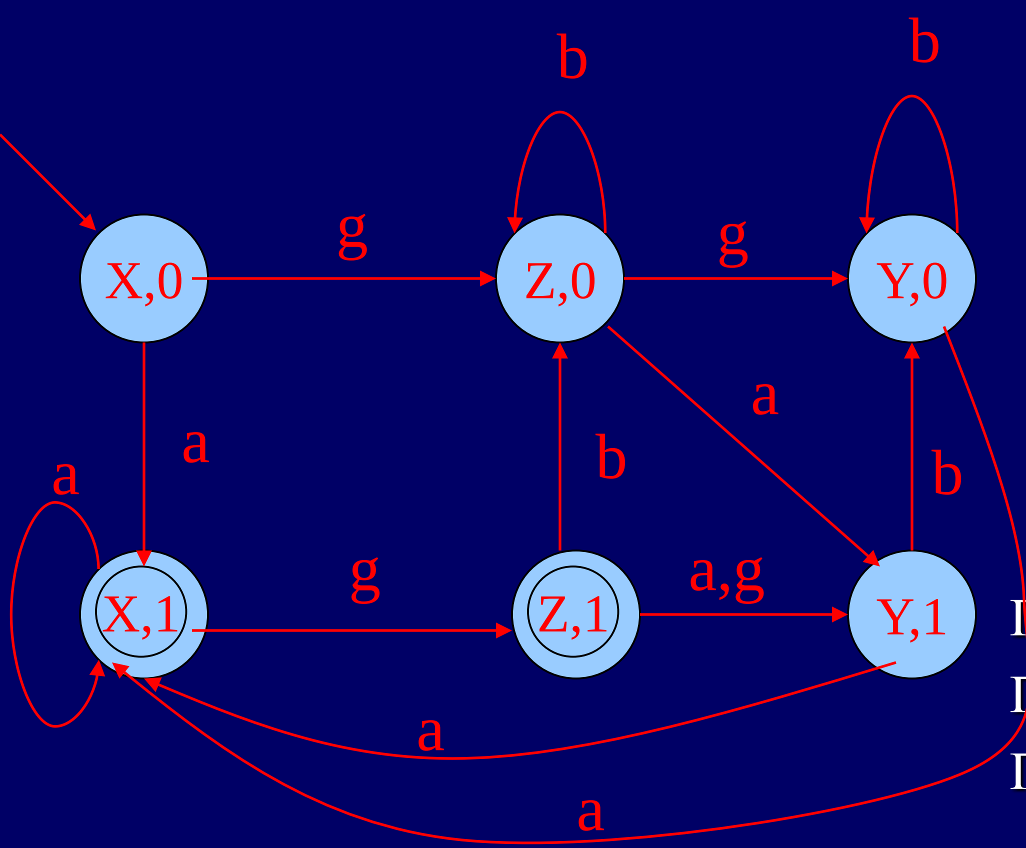
$S=\{s0,s1\}$, $E=\{a,b\}$, $f(s0,a) = s1$, $f(s0,b) = s0$, $f(s1,a)=s1$, $f(s1,b)=s0$, $\Gamma(s0) = \{a,b\}$, $\Gamma(s1) = \{a,b\}$, $S_m=\{s1\}$



Máquina de Estados

-Algumas Operações-

- Composição Paralela



S={(X,0),(X,1),(Y,0),(Y,1),(Z,0),(Z,1)},

E={a,b,g},

f((X,0),a) = (X,1), f((X,0),g) = (Z,0),

f((X,1),a) = (X,1), f((X,1),g) = (Z,1),

f((Y,0),a)=(X,1), f((Y,0),b)=(Y,0),

f((Y,1),a)=(X,1), f((Y,1),b)=(Y,0),

f((Z,0),b)=(Z,0),

f((Z,0),a)=f((Z,1),g)= f((Z,1),a)= (Y,1)

f((Z,1),b)= (Z,0)

Γ(X,0) = Γ(X,1) = {a,g},

Γ(Y,0) = Γ(Y,1) = {a,b},

Γ(Z,0) = Γ(Z,1) = {a,b,g}

S_m={(X,1),(Z,1)}

Máquina de Estados
-Algumas Operações-

- Composição Paralela

Se $E_1 = E_2$, todos os eventos de $SM_1||SM_2$ serão sincronizados

Se $E_1 \cap E_2 = \varnothing$, não se tem eventos sincronizados. Tem-se a concorrência, com nenhum sincronismo (*interleaving* dos eventos de SM_1 e SM_2)

Máquina de Estados
-Algumas Operações-

- Produto

$SM_1 = (S_1, E_1, f_1, \Gamma_1, s_0^1, S_m^1)$

$SM_2 = (S_2, E_2, f_2, \Gamma_2, s_0^2, S_m^2)$

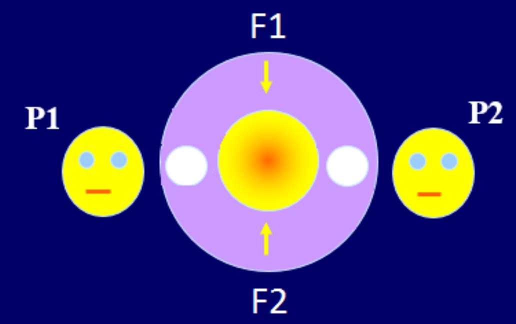
$SM_1 \times SM_2 = Rc(S_1 \times S_2, E_1 \cap E_2, f_{1\times 2}, \Gamma_{1\times 2}, (s_0^1, s_0^2), S_m^1 \times S_m^2)$

$$f_{1\times 2}((s_1, s_2), e) = \begin{cases} (f_1(s_1, e), f_2(s_2, e)) & \text{se } e \in \Gamma_1(s_1) \cap \Gamma_2(s_2) \\ \text{indefinido} & \text{caso contrário} \end{cases}$$

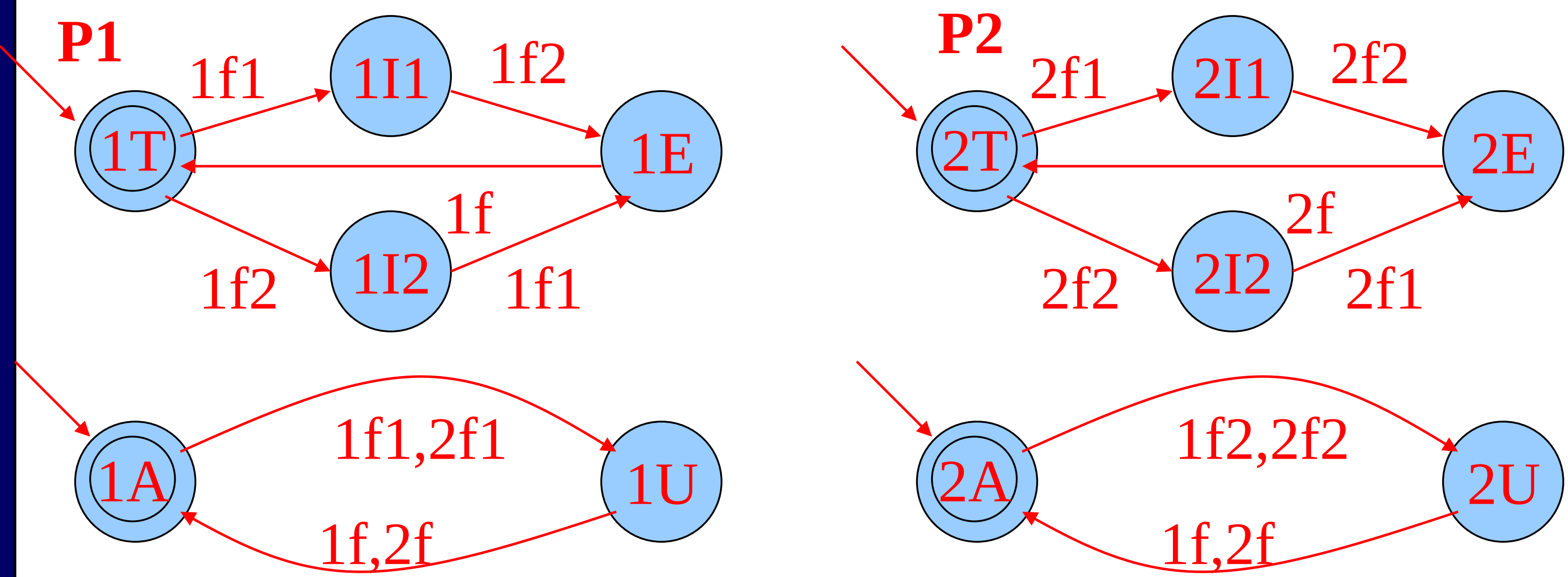
$\Gamma_{1\times 2}(s_1, s_2) = \Gamma_1(s_1) \cap \Gamma_2(s_2)$

Máquina de Estados

- Problema dos Filósofos -

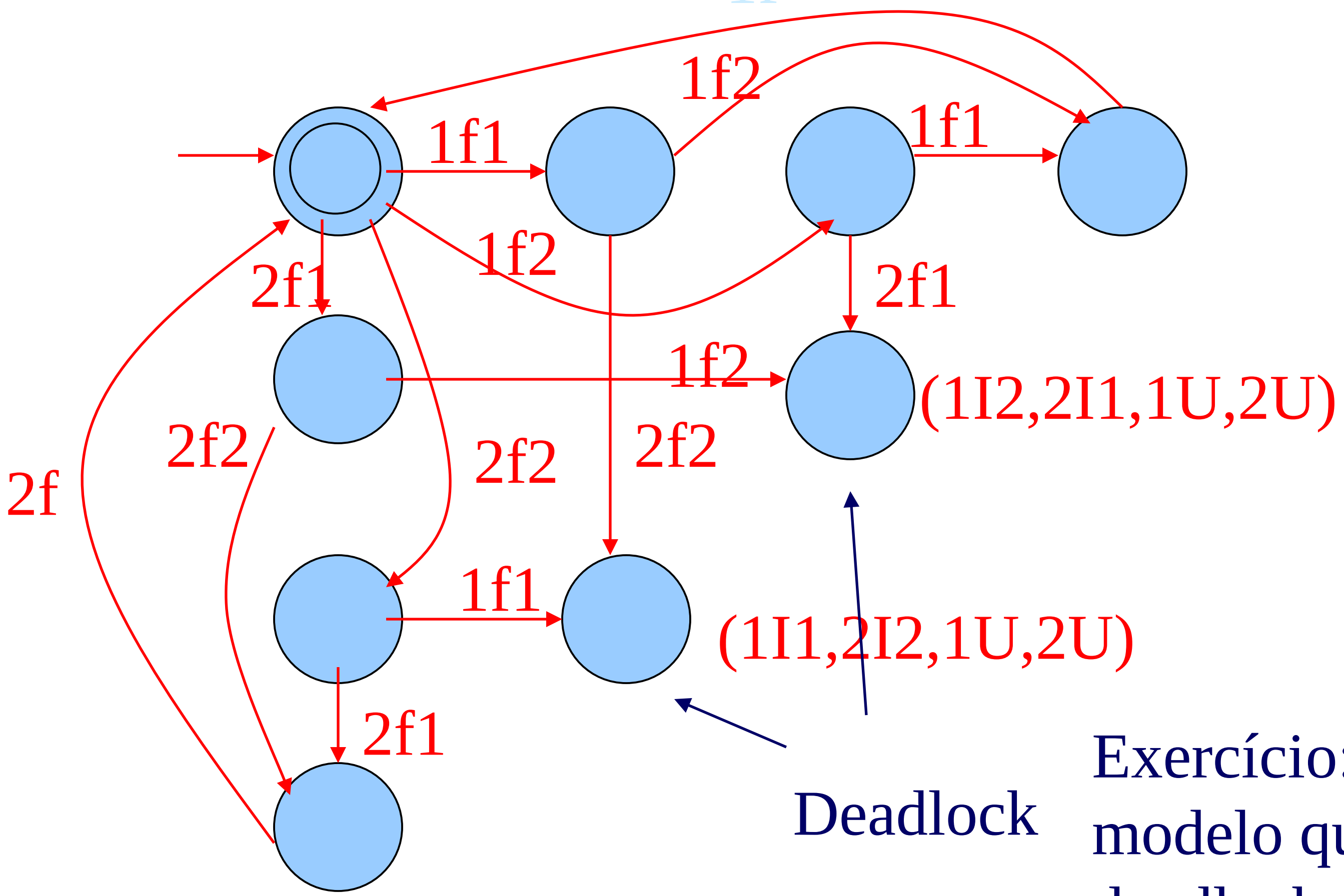


- Suponhamos dois Filósofos P1 e P2. Cada filósofo ou está pensando ou comendo. Os eventos *i*f*j* significam o filósofo *i* pega o garfo *j* e os evento *j*f significam o filósofo *j* libera os garfos.



Máquina de Estados

- Problema dos Filósofos -



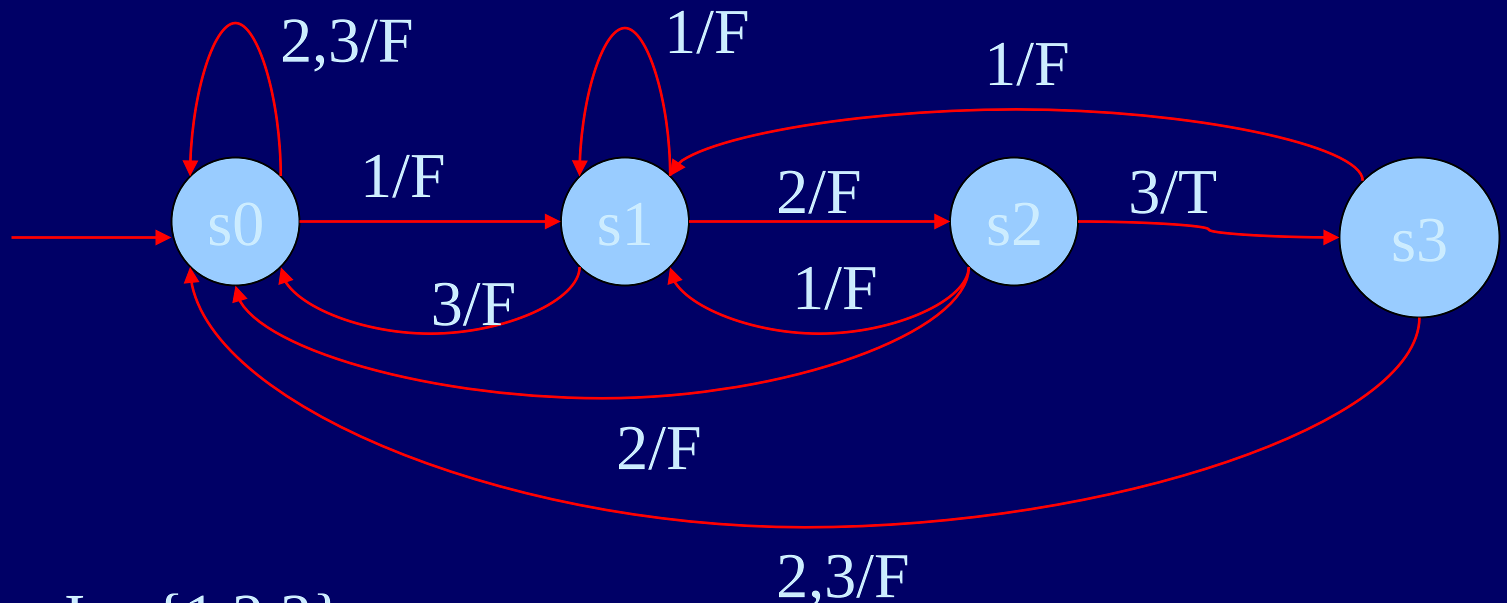
Exercício: construir um modelo que não tenha deadlock

Finite State Machine
Modelo Mealy

- SM= (S,I,O,f,h)
  - S – Conjunto de Estados
  - $s_0 \in S$– Estado inicial
  - I – Alfabeto de entrada
  - O – Alfabeto de saída
  - $f : S \times I \rightarrow S$ – Função de próximo estado
  - $h: S \times I \rightarrow O$ – Função de saída

Finite State Machine Modelo Mealy

- Detector de Seqüência 123

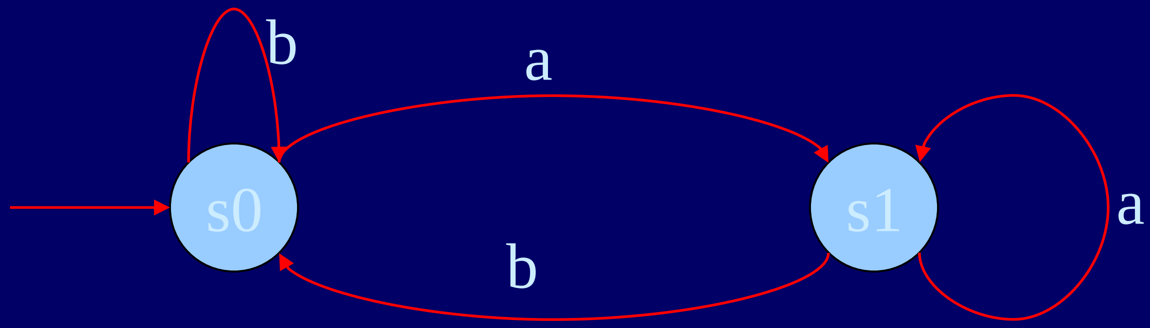


I = {1,2,3}
O={F,T}

Finite State Machine

- Linguagem Gerada

$$L(SM) = \{i_k \in I \mid f \text{ é definida}\}$$



L(SM) = {a, aa, ba, aaa, aba, ...}

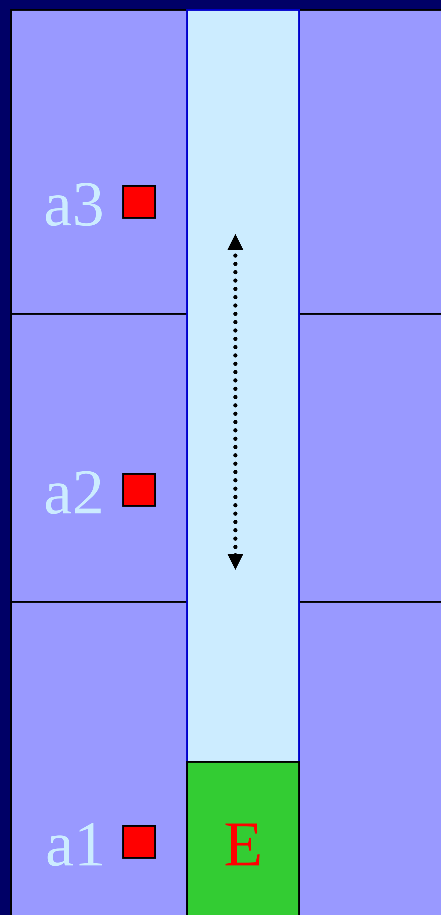
Duas Máquinas de Estados SM1 e SM2 são ditas equivalentes se L(SM1) = L(SM2)

Máquinas de Estados Finitos
Modelo Mealy

Edfícil

I={a1,a2,a3}

ANDAR DESEJADO

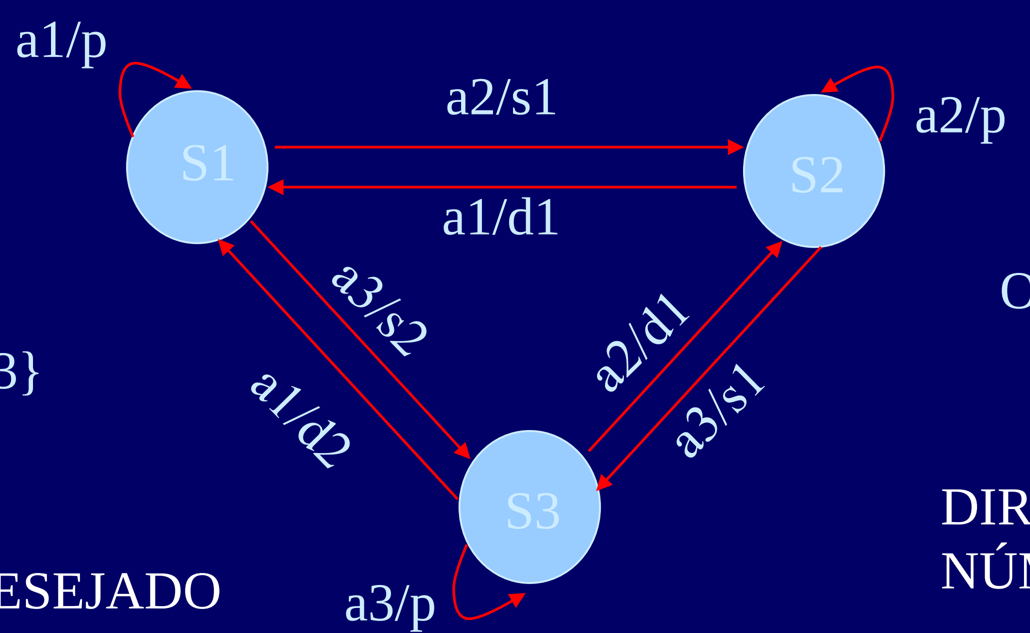


O={d2,d1,p,s1,s2}

DIREÇÃO E
NÚMERO

DE ANDARES

Máquinas de Estados Finitos
Modelo Mealy



I={a1,a2,a3}

ANDAR DESEJADO

O={d2,d1,p,s1,s2}

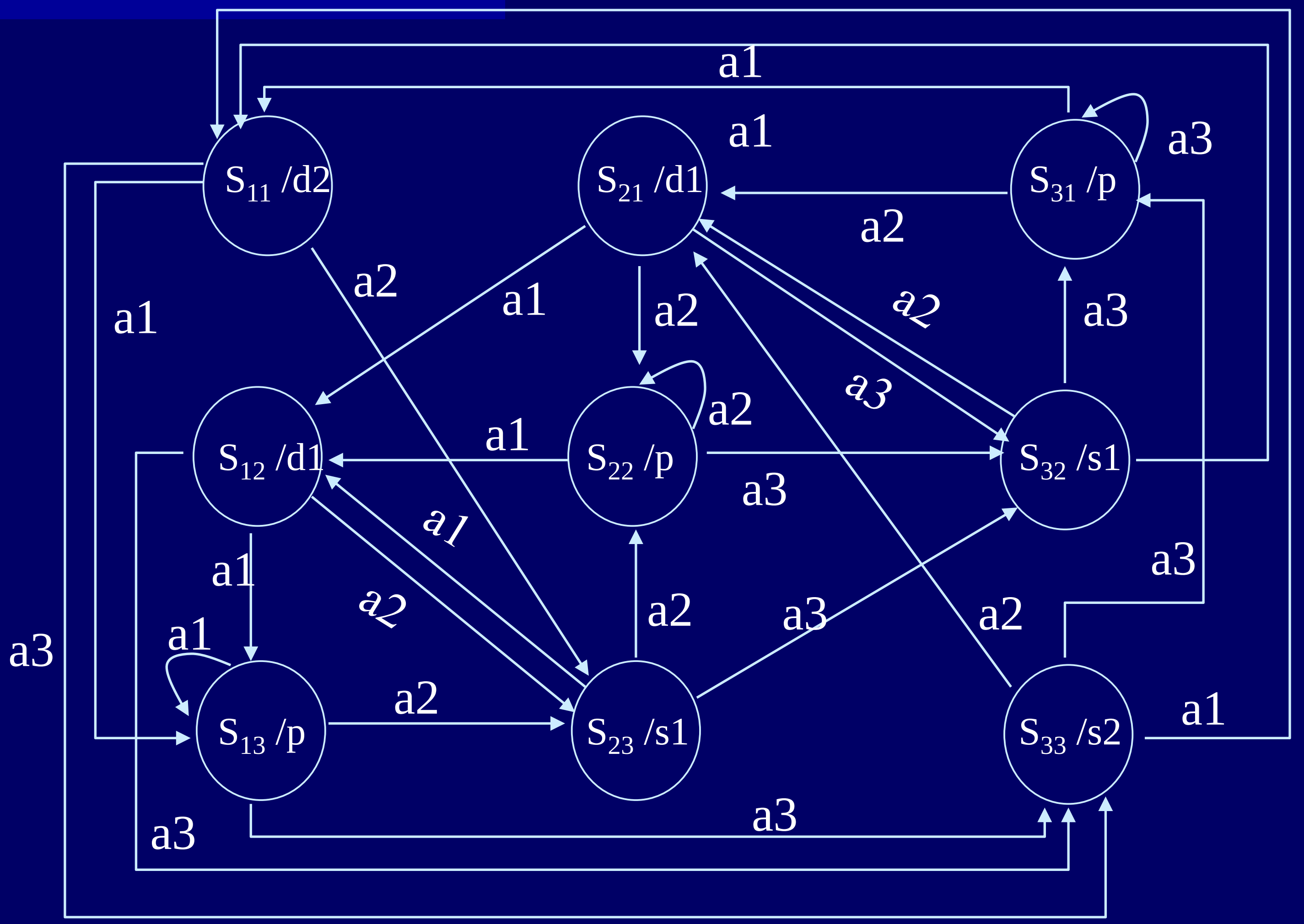
DIREÇÃO E NÚMERO

DE ANDARES

Máquinas de Estados Finitos Modelo Moore

- FSM= (S,I,O,f,h)
  - S – Conjunto de Estados
  - $s_0 \in S$– Estado inicial
  - I – Alfabeto de entrada
  - O – Alfabeto de saída
  - $f : S \times I \rightarrow S$ – Função de próximo estado
  - $h: S \rightarrow O$ – Função de saída

Máquinas de Estados Finitos
Modelo Moore



Máquinas de Estados Finitos

– Dificuldades na modelagem direta da concorrência.

- Criação de processos
- sincronização

– Impossibilidade de representação de sistemas a com número infinito de estados.

– A análise de propriedades interessantes são decidíveis

Sistema de Transição Rotulado

(Labeled Transition System)

Sistema de Transição Rotulado

- $TS = (S, s_0, L, tran)$
  - S – Conjunto de Estados
  - s_0 – Estado inicial
  - L – Conjunto de rótulos
  - $tran \subseteq S \times L \times S$, normalmente indicada por $s \xrightarrow{a} s'$

Finite State Process

(FSP)

Modelando Processos

Modelos serão descritos por *Labelled Transition Systems* LTS.

Serão descritos textualmente através *finite state processes* (FSP)

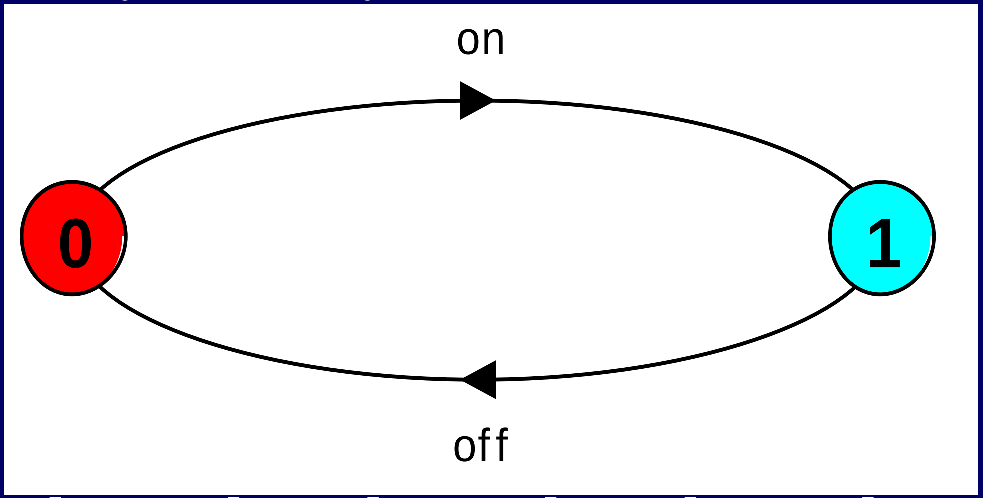
Ferramenta ***LTSA*** .

- LTS - forma gráfica
- FSP - forma algébrica

modelando processos

Um processo é a execução de programa sequencial.

Modelaremos utlizando LTS que muda de estado pela execução de ações atômicas.



a light switch **LTS**

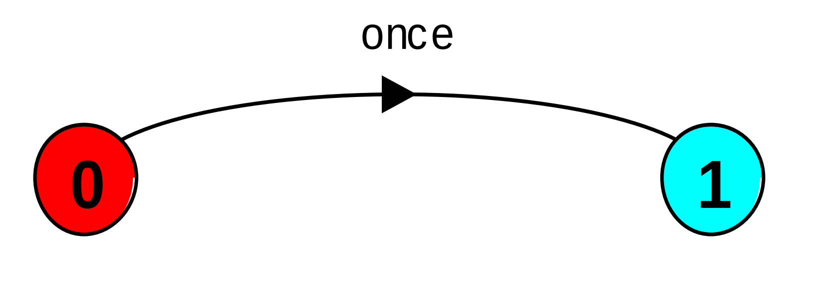
`on→off→on→off→on→off→ ……….`

a sequência de ações ou *trace*

FSP - Prefixação

Se **x** é uma ação e **P** um processo então **(x-> P)** descreve um processo que inicialmente executa a ação **x** e comporta-se exatamente como descrito pelo processo **P**.

```
ONESHOT = (once -> STOP).
```



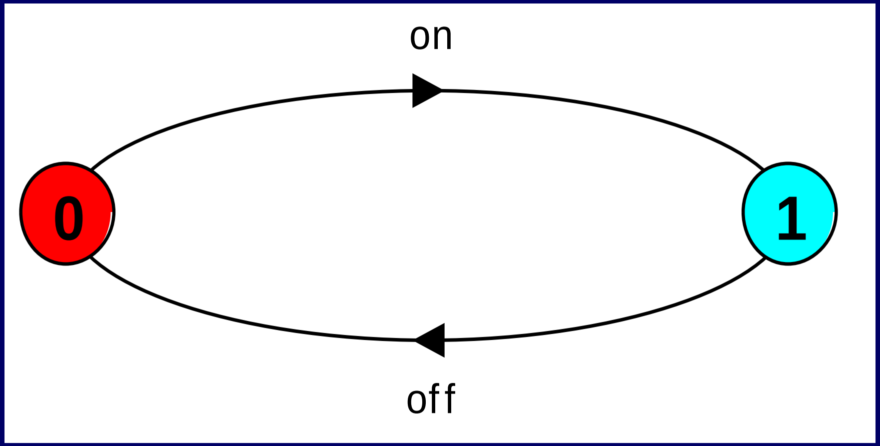
ONESHOT state machine

Convenção: ações começam com letras minúsculas e PROCESSOS com letras maiúsculas

FSP - Prefixação & Recursão

Comportamento Repetitivo - Usas-se recursão:

```
SWITCH = OFF,
OFF    = (on -> ON),
ON     = (off-> OFF).
```



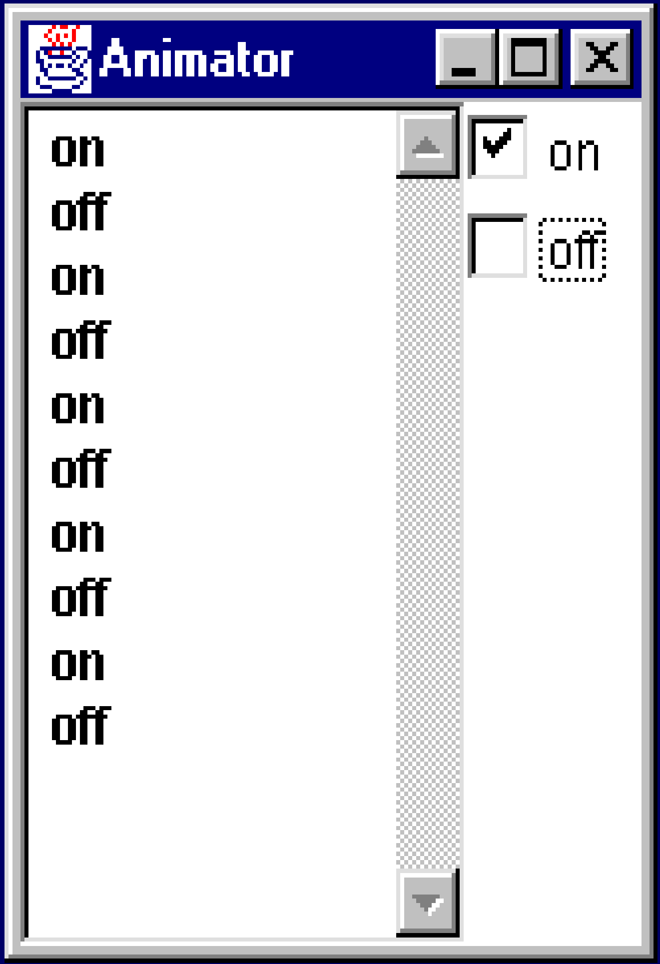
Forma mais sucinta:

```
SWITCH = OFF,
OFF    = (on ->(off->OFF)).
```

Outra vez:

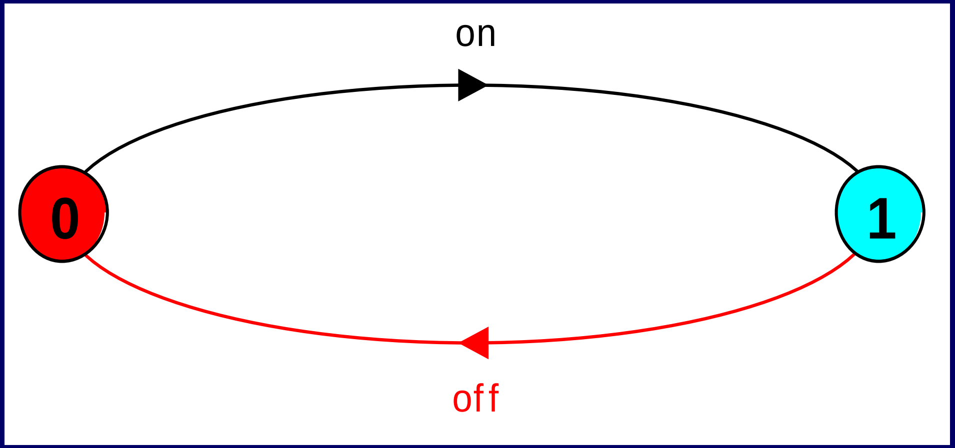
```
SWITCH = (on->off->SWITCH).
```

animação usando LTSA



O animador *LTSA* pode ser usado para produzir um *trace*.

Escolha das ações elegíveis.

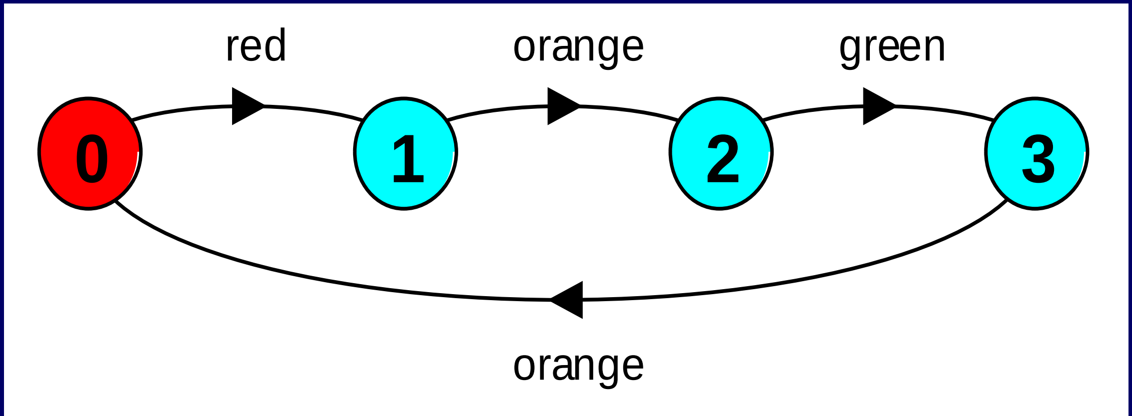


FSP - Prefixação

Modelo FSP de um semáforo :

```
TRAFFICLIGHT = (red->orange->green-
>orange    -> TRAFFICLIGHT).
```

LTS gerado utilizando *LTSA*:



Trace:

red→orange→green→orange→red→orange→green

…

FSP - Escolha

Se **x** e **y** são ações então **(x-> P | y-> Q)** descreve um processo que inicialmente executa

x or **y**. Após a primeira ação ter ocorrido, o comportamento subsequente é descrito po **P** se a primeira ação foi **x** e **Q** se foi **y**.

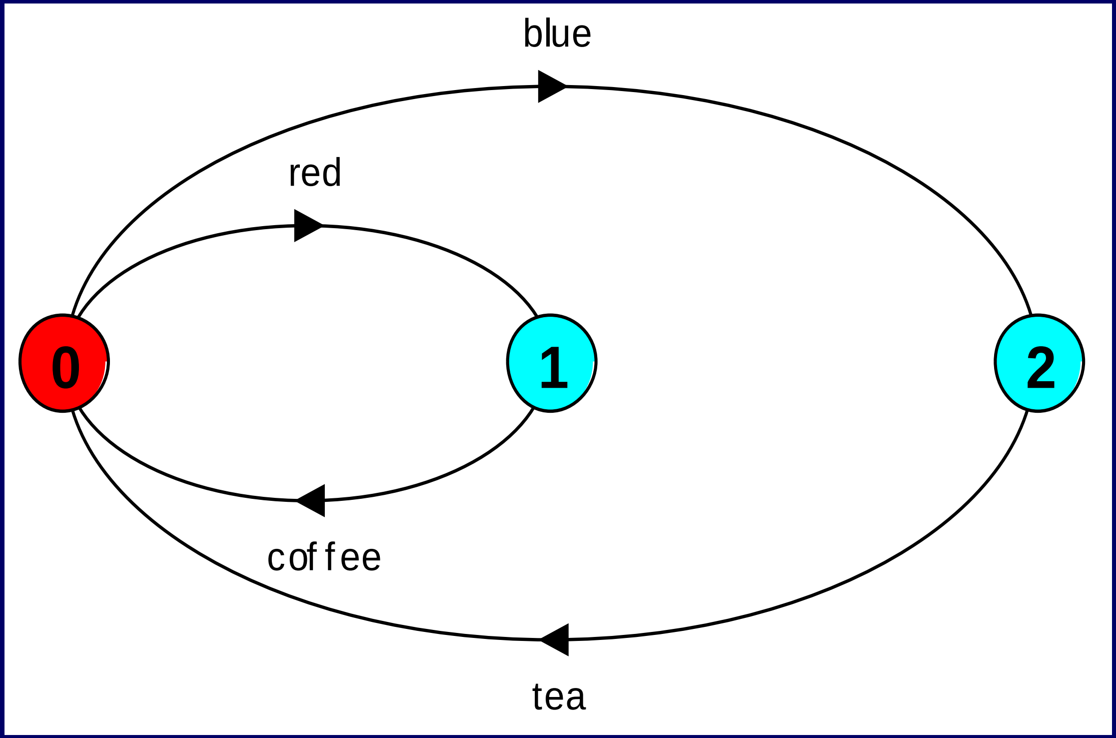
Quem o que fez a escolha ?

FSP - choice

Modelo FSP of uma máquina de venda :

```
DRINKS = (red->coffee->DRINKS
         |blue->tea->DRINKS
         ).
```

LTS gerado usando-se *LTSA*:

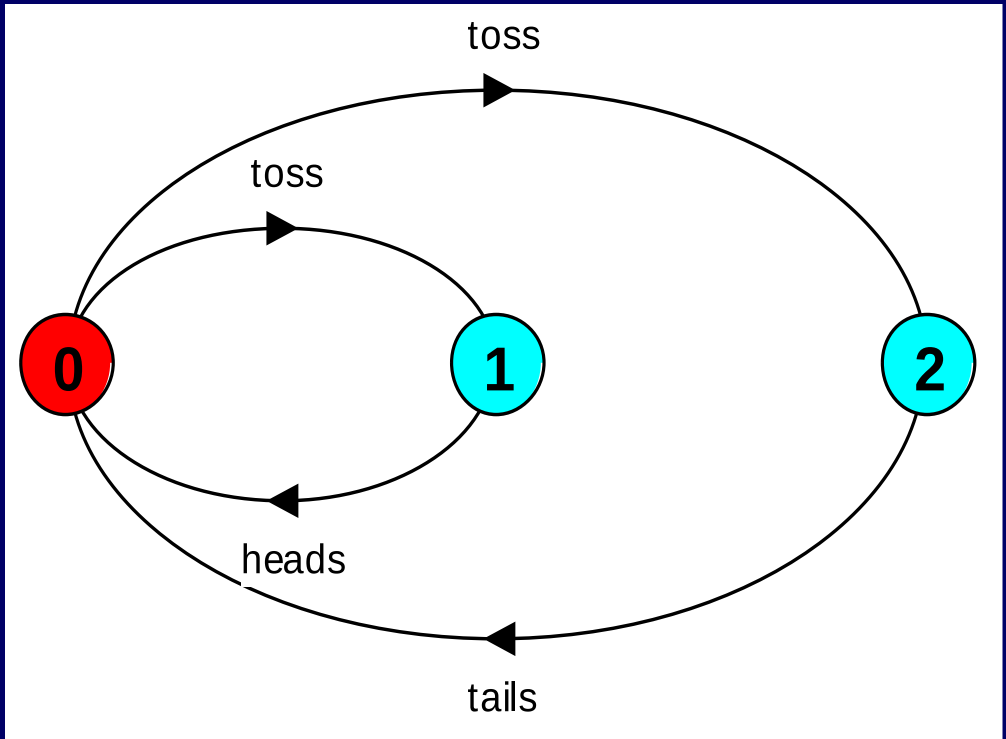


Quais são os *traces* possíveis?

Escolha Não-Determinística

Processo (x-> P | x -> Q) descreve um processo que executa **x** e então comporta-se como P ou Q.

```
COIN = (toss->HEADS|toss->TAILS),
HEADS= (heads->COIN),
TAILS= (tails->COIN).
```



Quais são os possíveis *traces*?

FSP - Processos e Ações Indexadas

Considere um buffer que recebe como entrada valores entre 0 e 3 e em seguida os fornece como saída:

```
BUFF = (in[i:0..3]->out[i]-> BUFF).
```

Equivale a

```
BUFF = (in[0]->out[0]->BUFF
       |in[1]->out[1]->BUFF
       |in[2]->out[2]->BUFF
       |in[3]->out[3]->BUFF
       ).
```

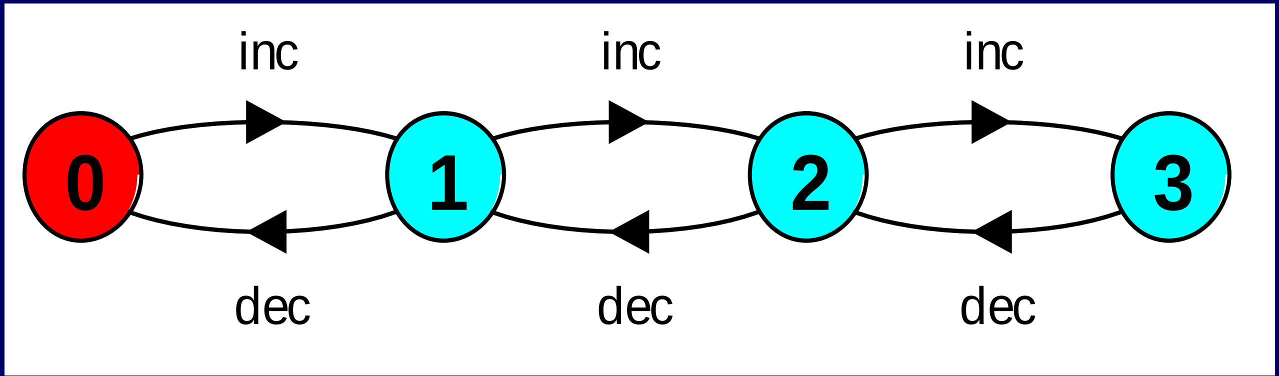
Ou através de parâmetros de processos com valor default:

```
BUFF(N=3) = (in[i:0..N]->out[i]-> BUFF).
```

FSP - Ações Guardadas

A escolha **(when B x -> P | y -> Q)** significa que quando a guarda **B** é verdadeira então as ações **x** e **y** são ambas elegíveis, caso contrário, se **B** is falso, então a ação **x** não pode ser escolhida.

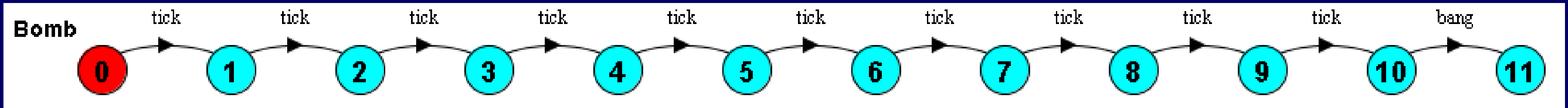
```
COUNT (N=3)      = COUNT[0],
COUNT[i:0..N] = (when(i<N) inc->COUNT[i+1]
                |when(i>0) dec->COUNT[i-1]
                ).
```



Finalização de Processos

O processo *deadlock* pode ser usado para finalizar um processo.

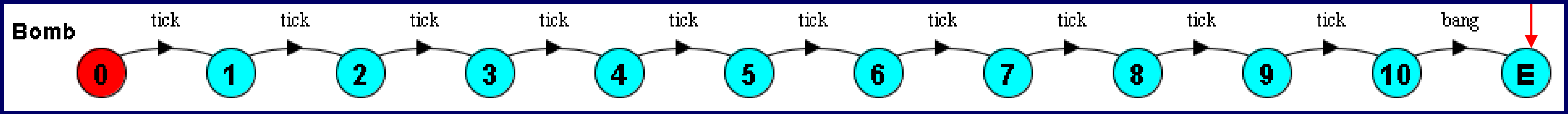
```
Bomb=(tick->tick->tick->tick->tick->tick->tick->tick->tick->tick->bang->STOP).
```



Finalização de Processos

Um processo END é um *deadlock* com nome especial E. Deve/pode ser usado para se especificar a finalização (adequada) de um processo.

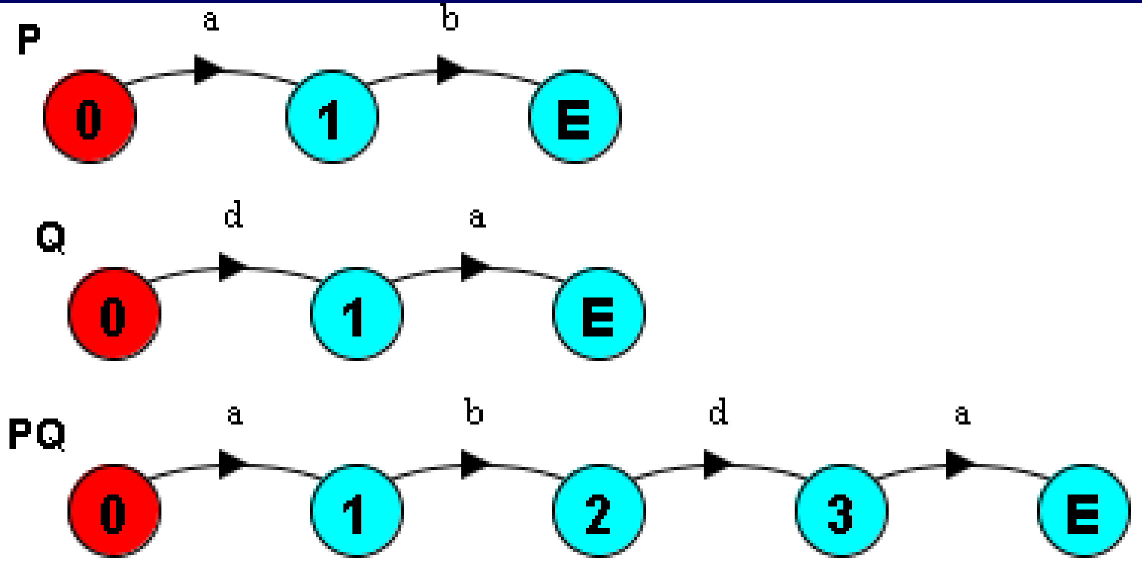
```
Bomb=(tick->tick->tick->tick->tick->tick->tick->tick->tick->tick->bang->END).
```



Composição Sequencial de Processos

Se P e Q são processos sequenciais, P;Q é a composição sequencial de P e Q.

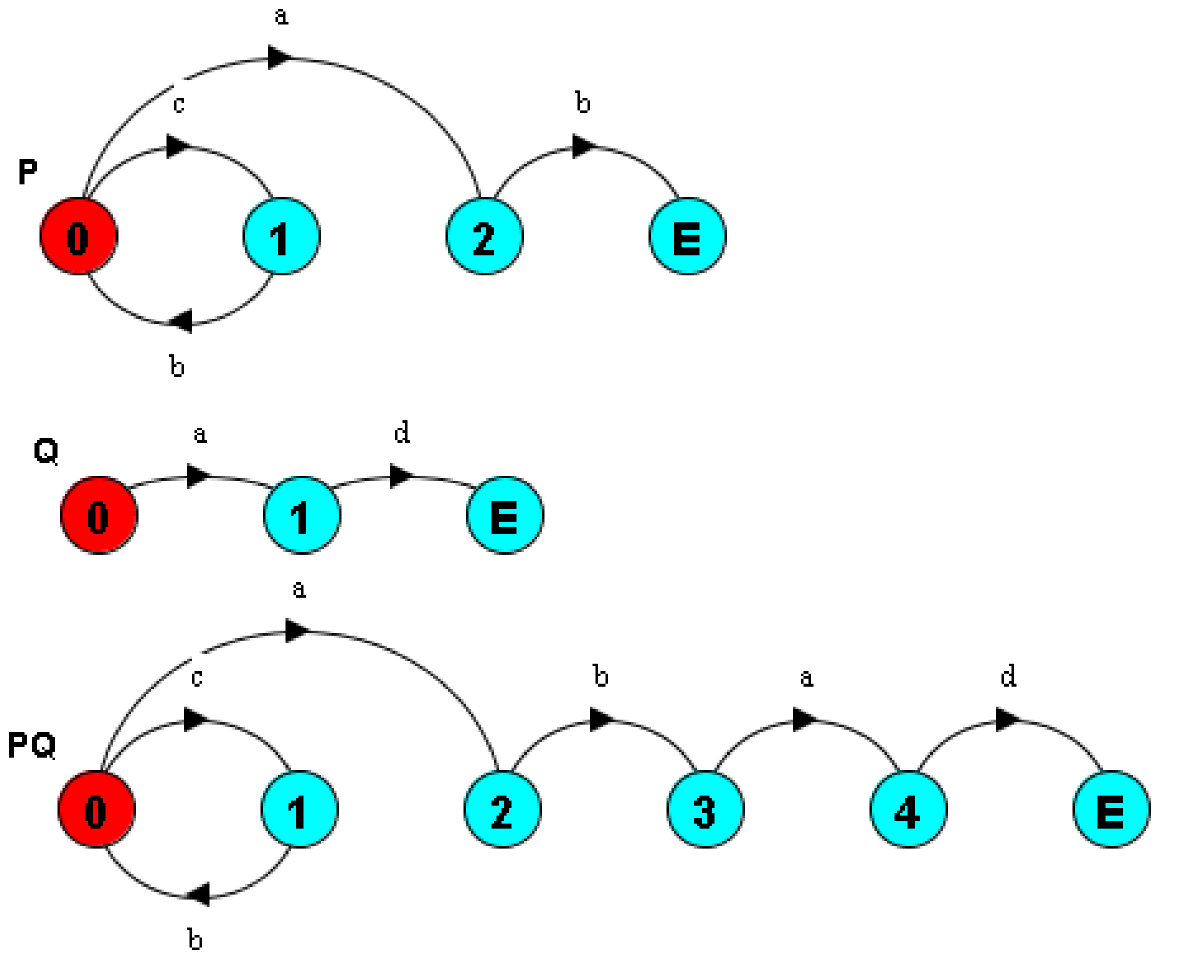
```
P=(a->b->END).
Q=(d->a->END).
PQ=P;Q;END.
```



Composição Sequencial de Processos

Outro exemplo:

```
P=(a->b->END
    |c->b->P).
Q=(a->d->END).
PQ=P;Q;END.
```



Composição Paralela *Interleaving* de Ações

Se P e Q são processos, então (P||Q) representa aa execução concorrente de P e Q. O operador || é o operador de composição paralela.

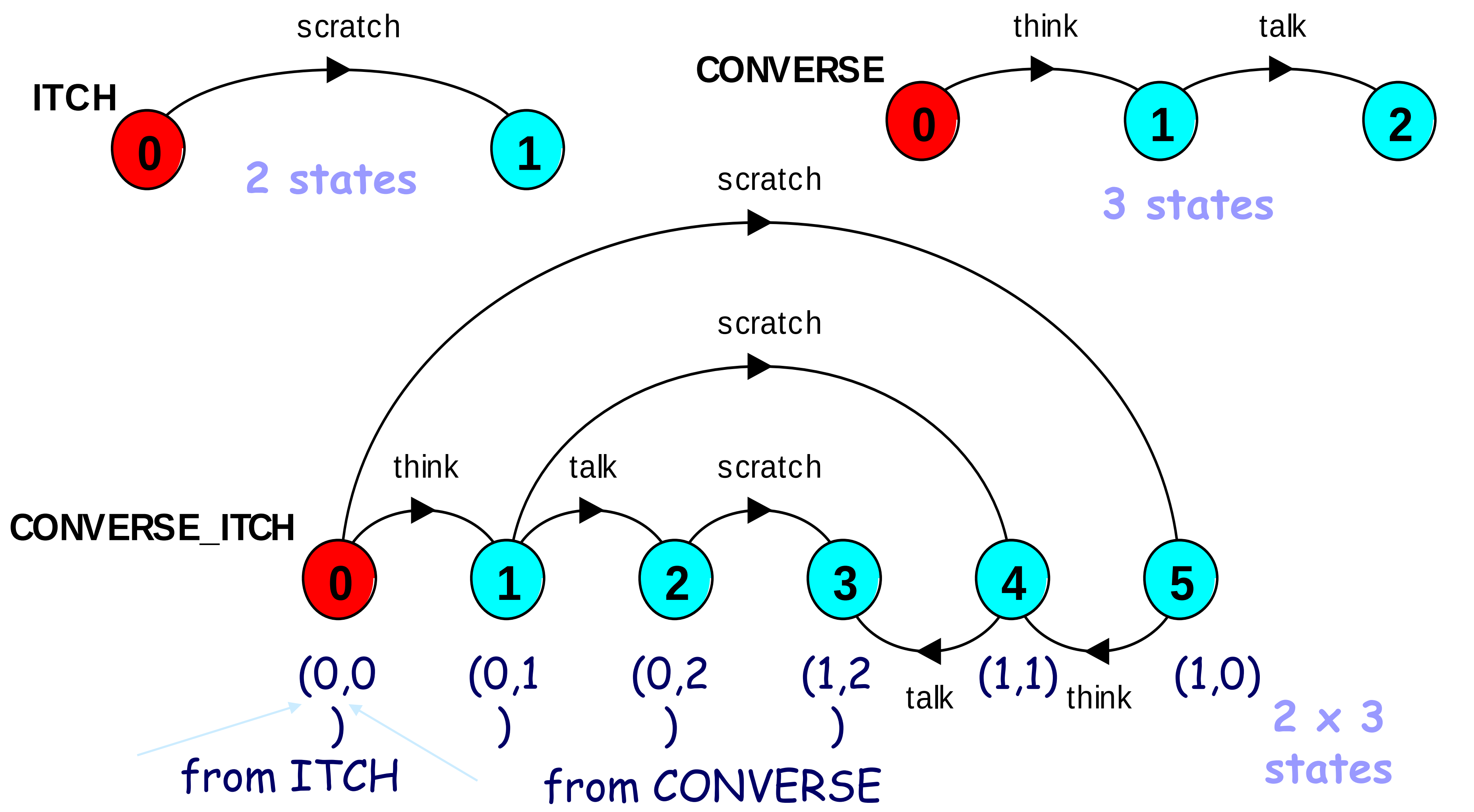
```
ITCH  = (scratch->STOP).
CONVERSE = (think->talk->STOP).

||CONVERSE_ITCH = (ITCH || CONVERSE).
```

Alfabeto disjunto

think→talk→scratch
think→scratch→talk
scratch→think→talk

Possíveis *traces* resultantes do *interleaving* de ações.

Composição Paralela
Interleaving de Ações



Composição Paralela – Leis Algébricas

Comutativa: `(P||Q) = (Q||P)`

Associativa: `(P||(Q||R)) = ((P||Q)||R)`
`= (P||Q||R).`

Clock radio:

```
CLOCK = (tick->CLOCK).
RADIO = (on->off->RADIO).

||CLOCK_RADIO = (CLOCK || RADIO).
```

LTS?   Traces?   Número de estados?

Composição Paralela *Interleaving* de Ações

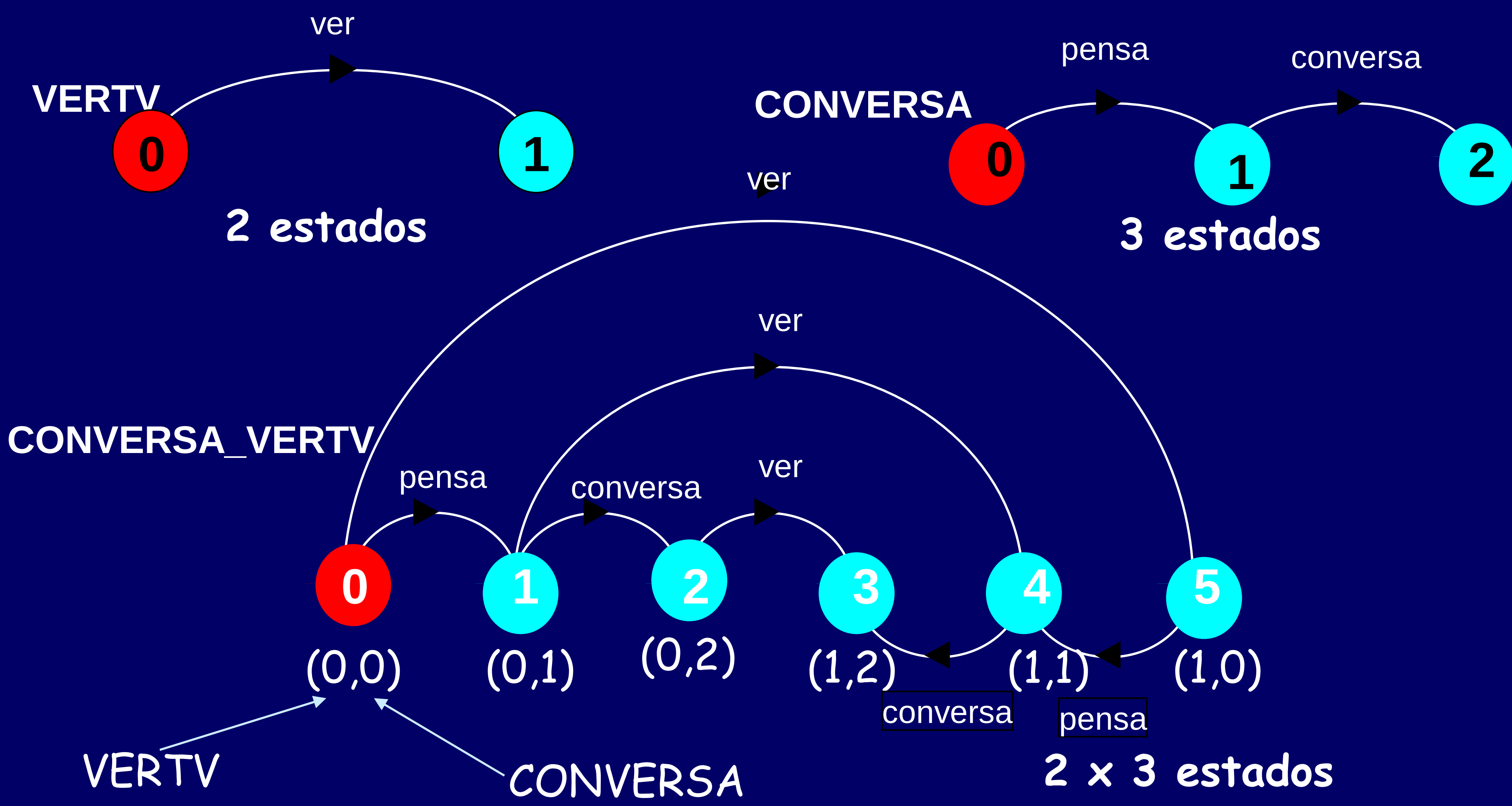
Se VERTV e CONVERSA são dois processos, então (VERTV||CONVERSA) representa a execução concorrente de VERTV e CONVERSA. O operador || é o operador de composição paralela.

```
VERTV    = (ver->STOP).
CONVERSA = (pensa->conversa->STOP).

||CONVERSA_VERTV = (VERTV || CONVERSA).
```

```
pensa→conversa→ver
pensa→ver→conversa
ver→pensa→conversa
```

Os *traces* possíveis são resultados do *interleaving* de ações.

Composição Paralela
Interleaving de Ações



Modelando Interações
Ações Compartilhadas

Se processoes em uma composição têm ações em comum, estas ações são ditas compartilhadas.

Ações compartilhadas modelam as interações entre processos.

Enquanto ações não compartilhadas podem ser arbitrariamente *interleaved* , ações compartilhadas devem ser executadas ao mesmo tempo por todos os processos.

```
MAKER = (make->ready->MAKER).
USER  = (ready->use->USER).

||MAKER_USER = (MAKER || USER).
```

MAKER sincroniza-se com **USER** quando **ready**.

Modelando Interações
Ações Compartilhadas

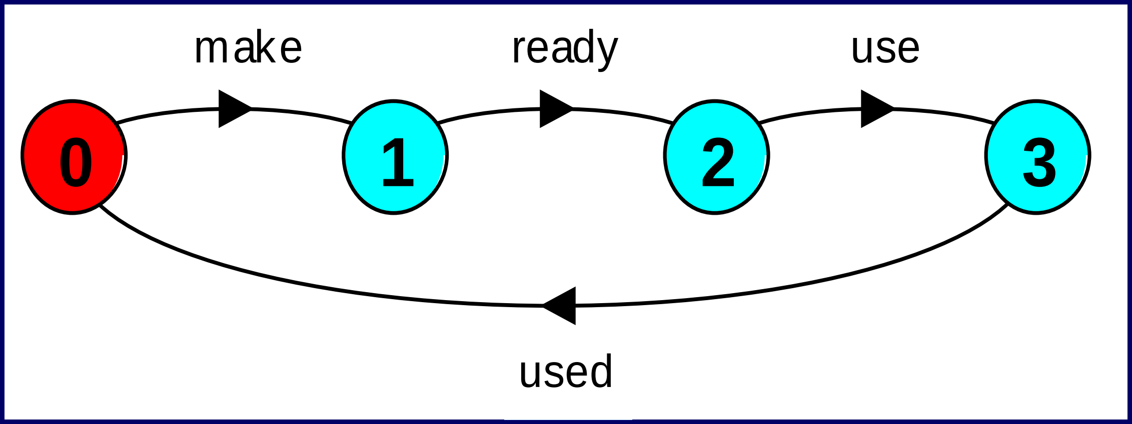
```
MAKERv2 = (make->ready->used->MAKERv2).
USERv2  = (ready->use->used ->USERv2).
```

3 estados

3 estados

```
||MAKER_USERv2 = (MAKERv2 || USERv2).
```

3 x 3 estados?



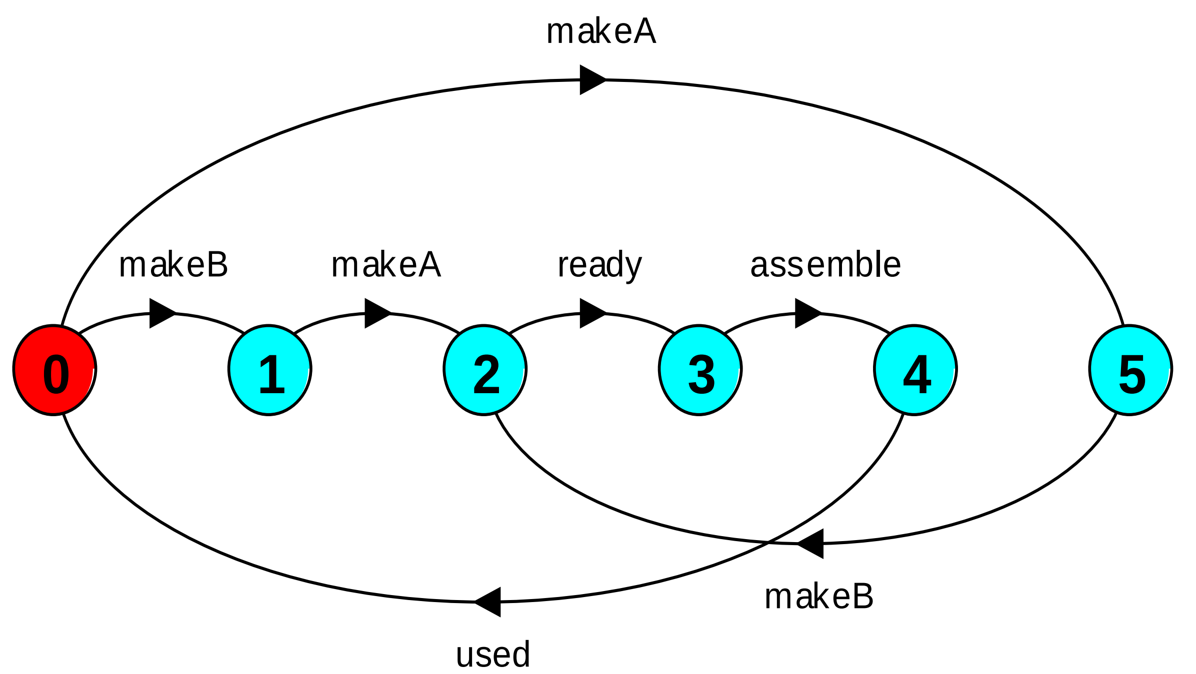
4 estados

Interação restringe o comportamento global

Modelando Interações Múltiplos Processos

Sincronização Múltipla:

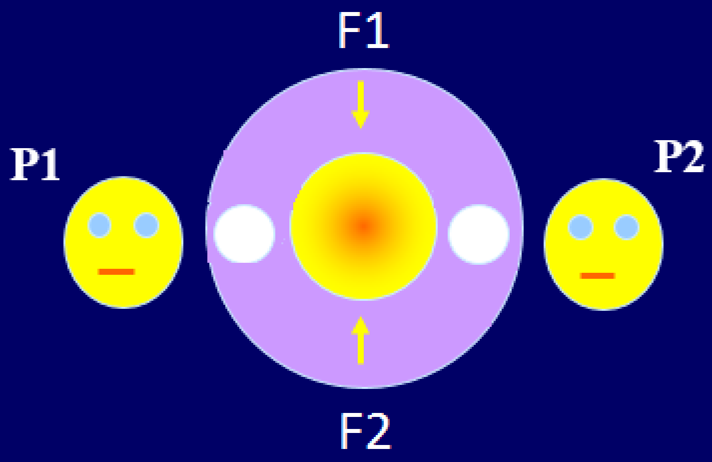
```
MAKE_A   = (makeA->ready->used->MAKE_A).
MAKE_B   = (makeB->ready->used->MAKE_B).
ASSEMBLE = (ready->assemble->used->ASSEMBLE).

||FACTORY = (MAKE_A || MAKE_B || ASSEMBLE).
```



Modelando Interações Múltiplos Processos

Sincronização Múltipla:

Jantar dos Filósofos




```
Trace to DEADLOCK:
        lf1
        sf2
```

Suponhamos dois Filósofos P1 e P2.

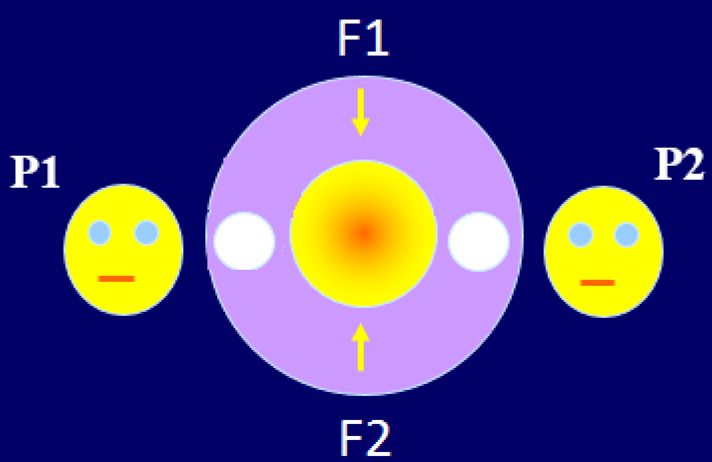
Cada filósofo ou está pensando ou comendo. Os eventos *ifj* significam o filósofo *i* pega o garfo *j* e os evento *j*f significam o filósofo *j* libera os garfos.

```
P1=(lf1->lf2->lf->P1).
P2=(sf2->sf1->sf->P2).
F1=(lf1->lf->F1
   |sf1->sf->F1).
F2=(lf2->lf->F2
   |sf2->sf->F2).
||PD=(P1||P2||F1||F2).
```

Modelando Interações Múltiplos Processos

Sincronização Múltipla:

Jantar dos Filósofos sem *deadlock*




```
No deadlocks/errors
```

Suponhamos dois Filósofos P1 e P2.

Cada filósofo ou está pensando ou comendo. Os eventos *ifj* significam o filósofo *i* pega o garfo *j* e os evento *j*f significam o filósofo *j* libera os garfos.

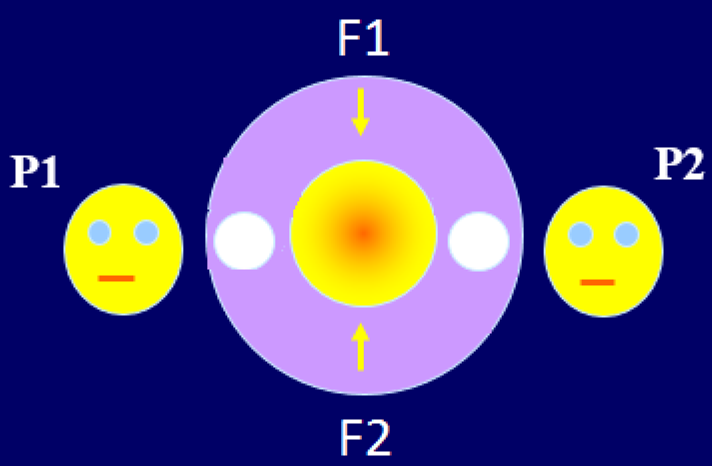
Solução obtida através da modificação do comportamento de P2.

```
P1=(lf1->lf2->lf->P1).
P2=(sf1->sf2->sf->P2).
F1=(lf1->lf->F1
   |sf1->sf->F1).
F2=(lf2->lf->F2
   |sf2->sf->F2).
||PD=(P1||P2||F1||F2).
```

Modelando Interações Múltiplos Processos

Sincronização Múltipla:

Jantar dos Filósofos sem *deadlock*




```
No deadlocks/errors
```

Suponhamos dois Filósofos P1 e P2.

Cada filósofo ou está pensando ou comendo. Os eventos *ifj* significam o filósofo *i* pega o garfo *j* e os evento *jf* significam o filósofo *j* libera os garfos.

Solução obtida pela inclusão de um controlador.

```
P1=(lf1->lf2->lf->P1).
P2=(sf2->sf1->sf->P2).
F1=(lf1->lf->F1
   |sf1->sf->F1).
F2=(lf2->lf->F2
   |sf2->sf->F2).
C=(lf1->lf2->C
  |sf2->sf1->C).
||DFPD=(C||P1||P2||F1||F2).
```

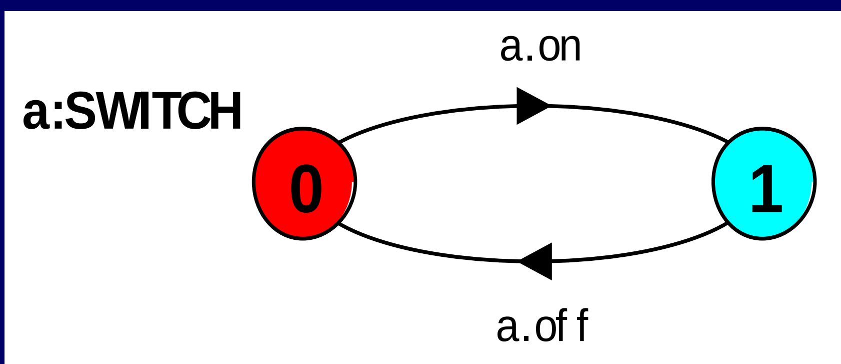
Nomeação de Processo

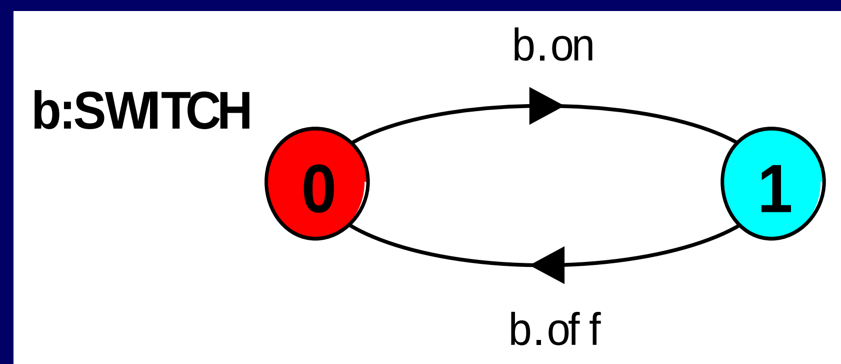
a:P prefixa cada rótulo associado a uma ação do alfabeto P com a.

```
SWITCH = (on->off->SWITCH).
```

Duas **instâncias** de switch process:

```
||TWO_SWITCH = (a:SWITCH || b:SWITCH).
```





Um array de **instâncias** do processo switch:

```
||SWITCHES(N=3) = (forall[i:1..N] s[i]:SWITCH).
```

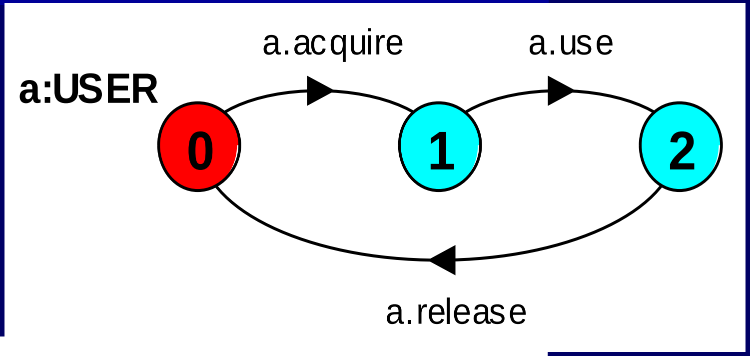
Nomeação de processo por um conjunto de rótulos prefixos

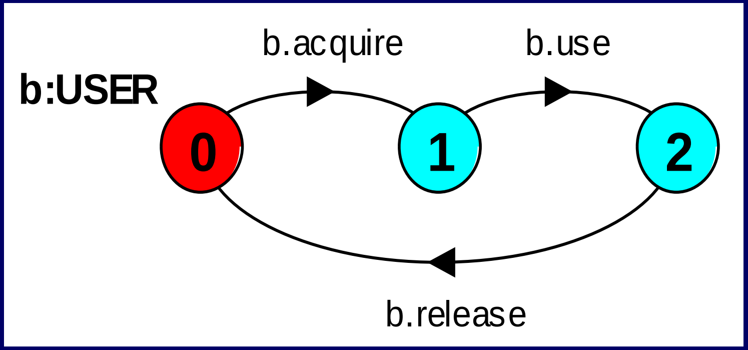
{a1,..,ax}::P substitui todo rótulo associado a uma ação no alfabeto de P com os rótulos a1.n,…,ax.n.

Posteriormente, toda ação (n->X) na definição de P será substituída pelas transições ({a1.n,…,ax.n} ->X).

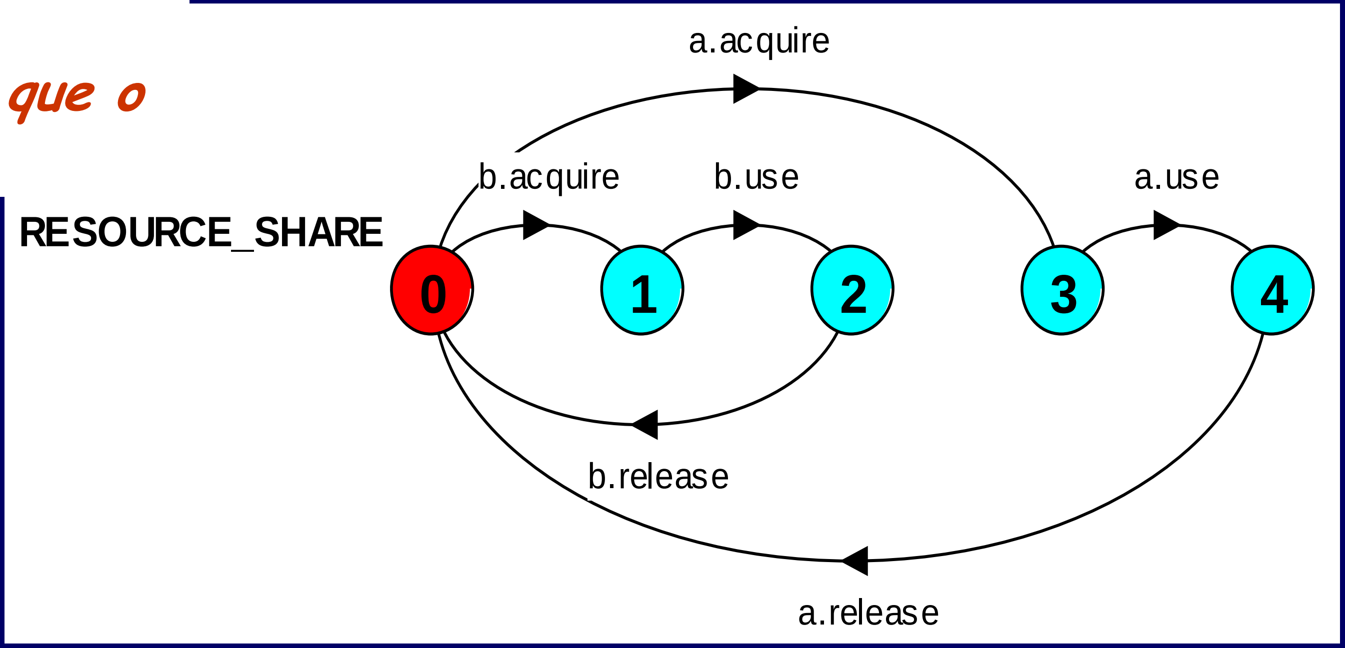
```
RESOURCE = (acquire->release->RESOURCE).
USER = (acquire->use->release->USER).

||RESOURCE_SHARE = (a:USER || b:USER
                    || {a,b}::RESOURCE).
```

Rótulos Prefixados a Processos



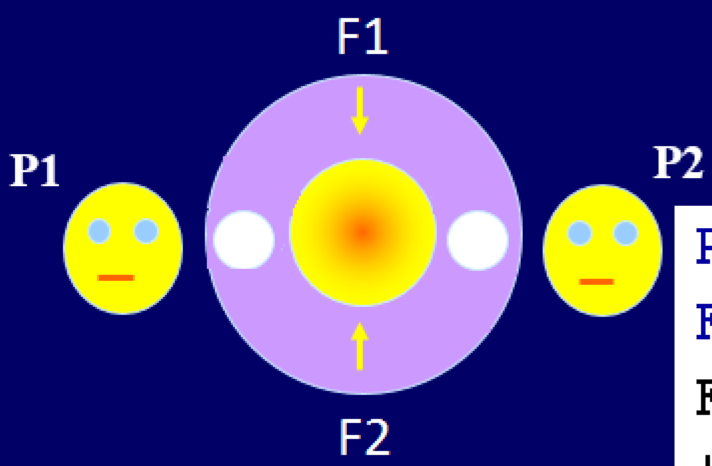


Como se garante que um usuário solicita um recurso é o mesmo que o libera ?



Rótulos Prefixados a Processos

Jantar dos Filósofos

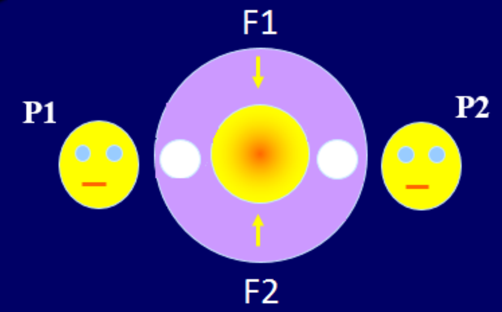



```
P=(f1->f2->f->P).
F1=(f1->f->F1).
F2=(f2->f->F2).
||PD=(a:P||b:P||{a,b}::F1||{a,b}::F2).
```

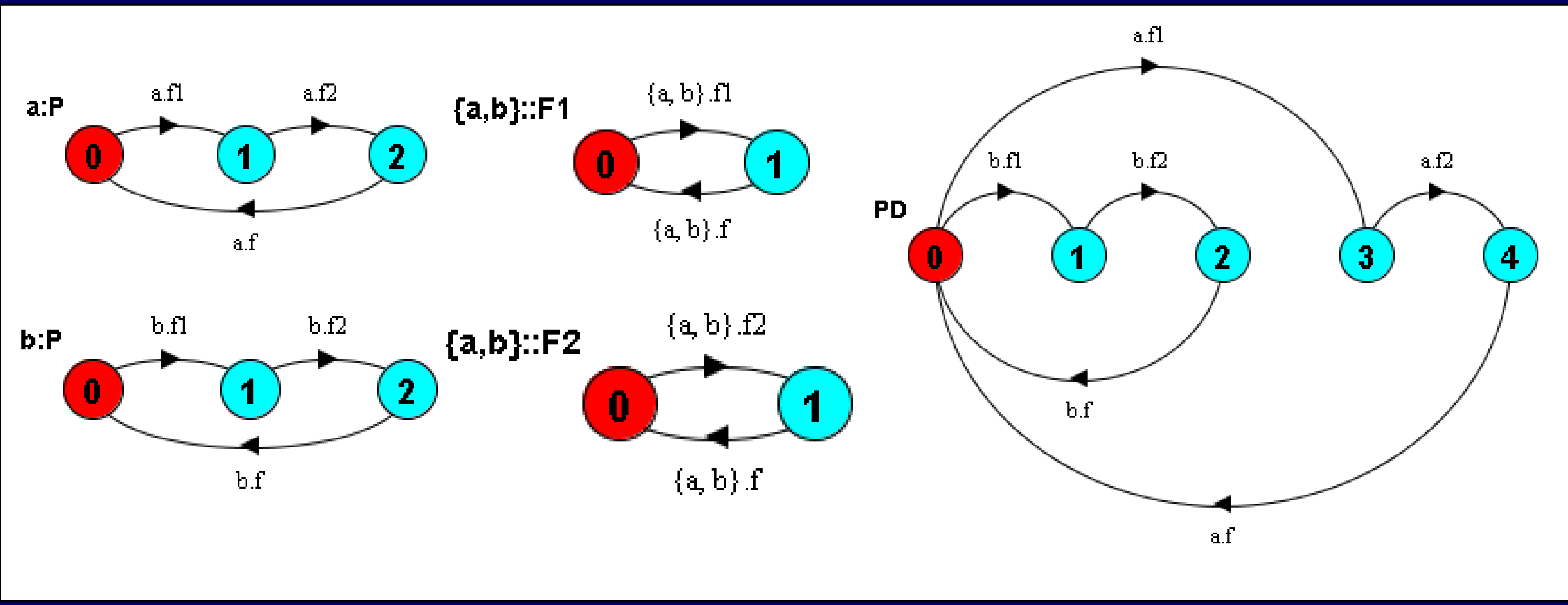
No deadlocks/errors

Rótulos Prefixados a Processos

Jantar dos Filósofos



No deadlocks/errors



Renomeação de Ações

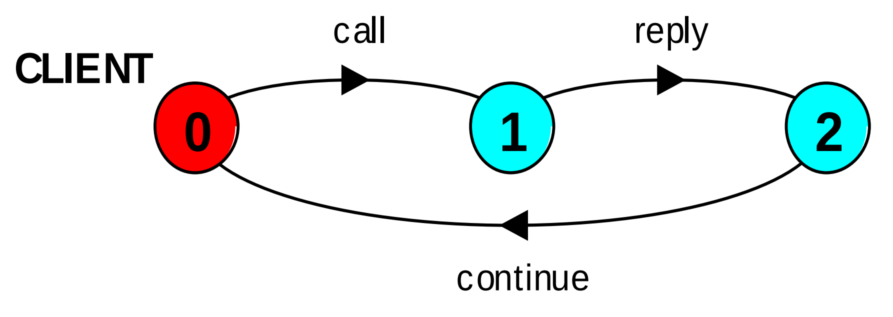
Funções de renomeação são usadas para mudar os nomes das ações. A forma geral é:
/{*newlabel_1*/*oldlabel_*1,... *newlabel_*n/*oldlabel_*n}.

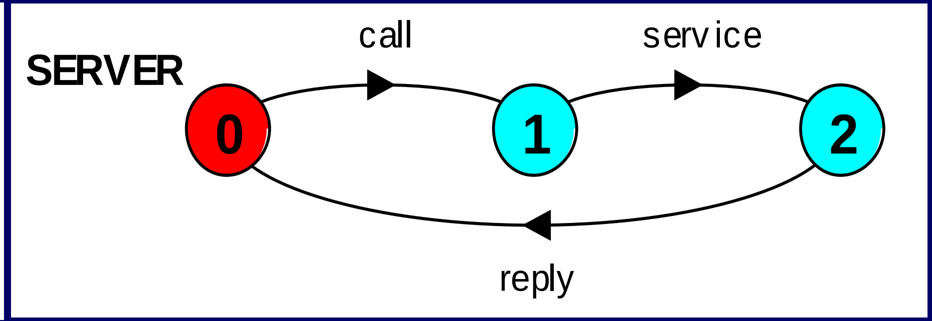
O renomeação garante que processos compostos se sincronizarão em uma ação em particular.

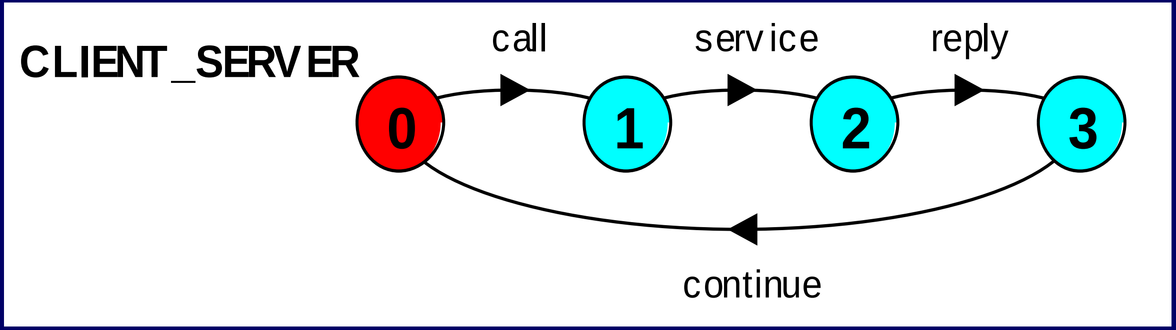
```
CLIENT = (call->wait->continue->CLIENT).
SERVER = (request->service->reply->SERVER).
```

Renomeação de Ações

```
||CLIENT_SERVER = (CLIENT || SERVER)
                  /{call/request, reply/wait}.
```







Tornando Internas (hiding) as Ações- Abstração para Redução de Complexidade

Quando apliado a um processo P, o operador hiding \{a1..ax} remove os nomes das ações a1..ax do alfabeto de P e torna estas ações "*silent*". Estas ações são rotuladas por tau. Ações *silent* em processos diferentes não são compartilhadas.

Algumas vezes é importante mostra as ações que não são *silents*.

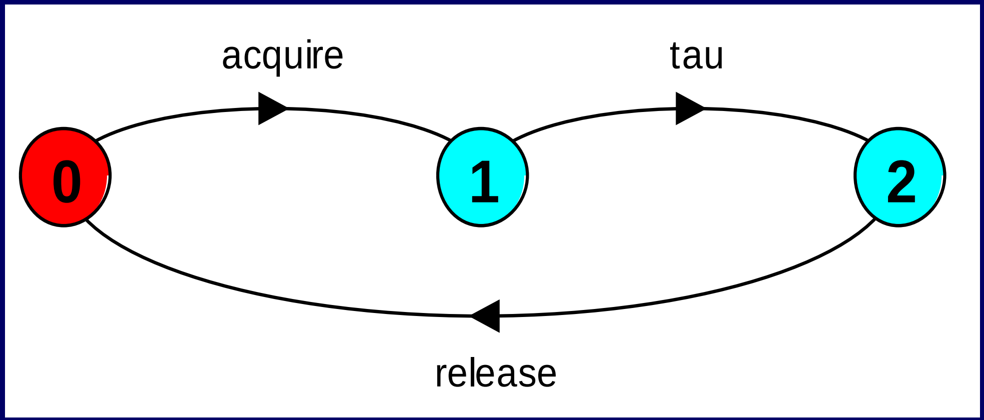
Quando aplicado ao processo P, o operador de interface @{a1..ax} esconde todas as ações exceto as presentes no conjunto a1..ax.

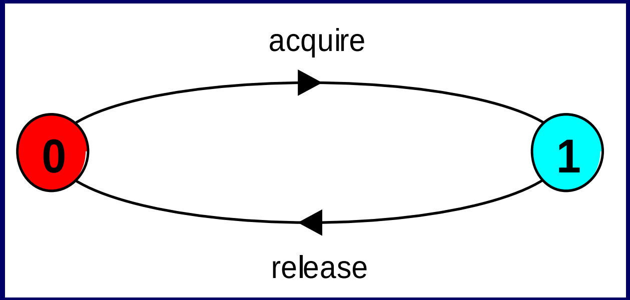
Tornando Internas (hiding) as Ações

As seguintes definições são equivalentes:

```
USER = (acquire->use->release->USER)
       \{use}.

USER = (acquire->use->release->USER)
         @{acquire,release}.
```

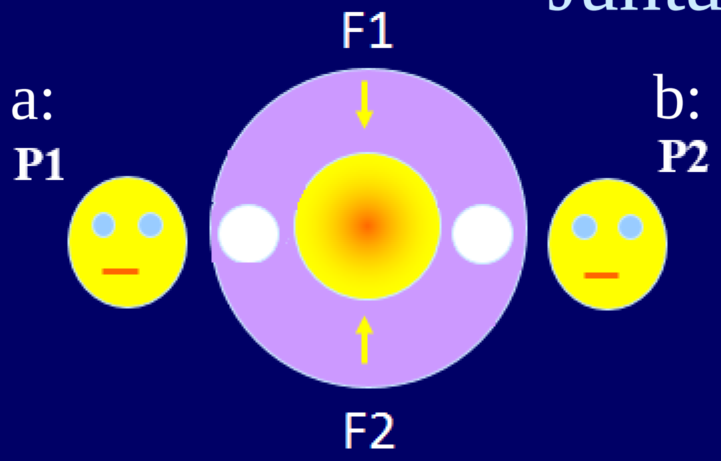
A minimização remove as ações *silents* produzindo um LTS com ações obesrváveis.





Tornando Internas (hiding) as Ações

Jantar dos Filósofos



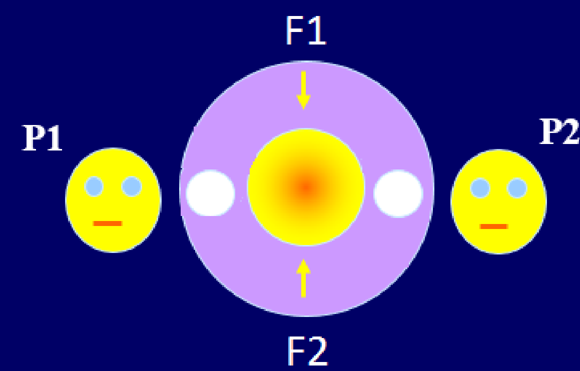
A especificação de P explicita as ações *eating* (de fato, três vezes!)

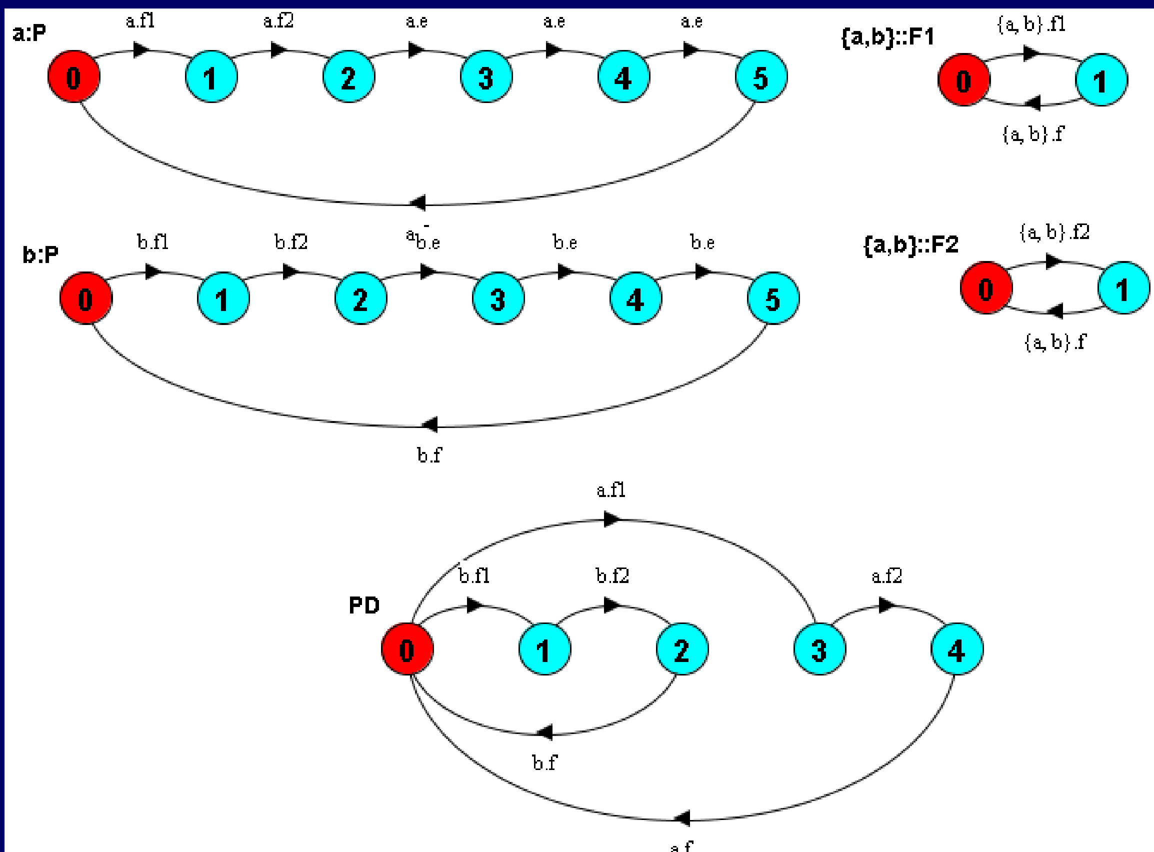
Contudo, se não se desejar gerar o LTS com essas ações, pode-se abstrair e torná-las "internas"/ "indistinguíveis".

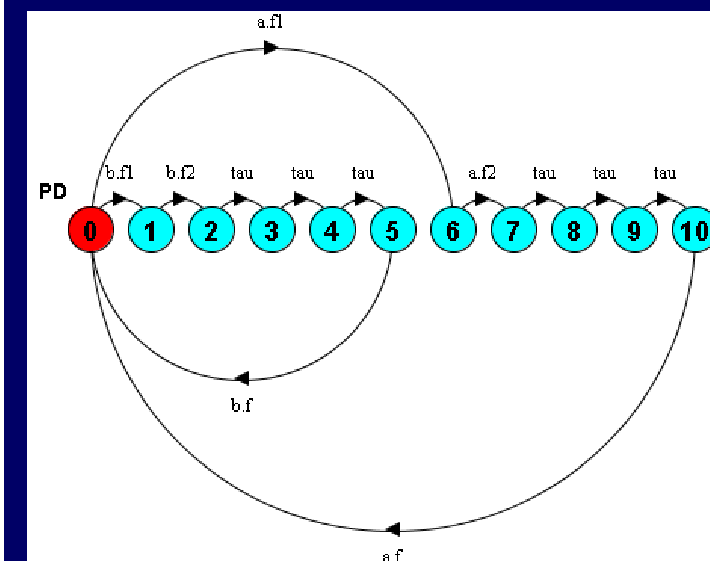
```
P=(f1->f2->e->e->e->f->P).
F1=(f1->f->F1).
F2=(f2->f->F2).
||PD=(a:P||b:P||{a,b}::F1||{a,b}::F2)\{a.e,b.e}.
```

Tornando Internas (hiding) as Ações

Jantar dos Filósofos





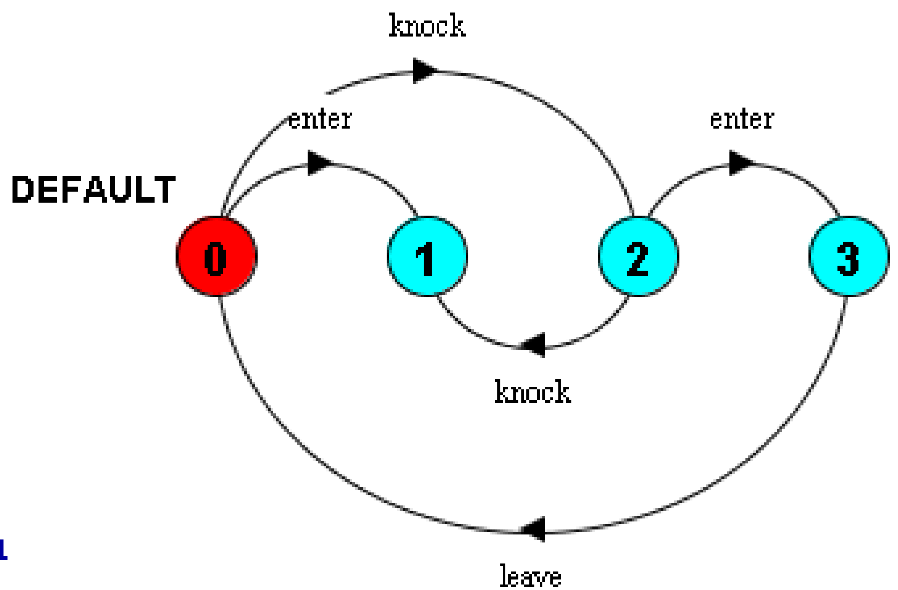


Safety
Alerta especificado explicitamente

Considere a situação em que uma pessoa para entrar numa sala (*room*), deve bater (*knock*) antes de entrar. Depois de entrar na sala, a pessoa pode sair (*leave*).

Considera-se inapropriado entrar na sala sem bater, tampouco bater duas vezes na porta.

Gostaríamos de modelar este processo e gerar "um alerta" quando um comportamento inapropriado ocorrer.

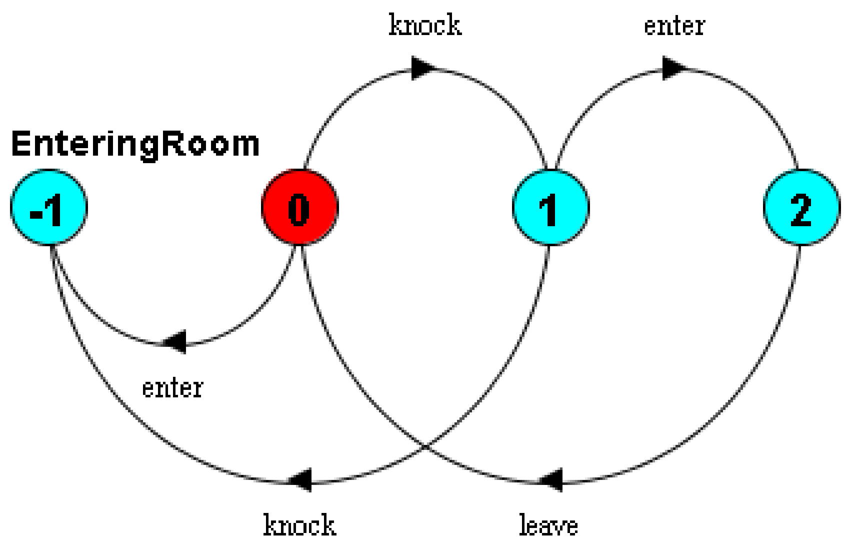



```
EnteringRoom=(knock->Enter
             |enter->STOP),
Enter=(enter->leave->EnteringRoom
      |knock->STOP).
```

Safety
Alerta especificado explicitamente

Um processo ERROR é um *deadlock* com nome especial -1

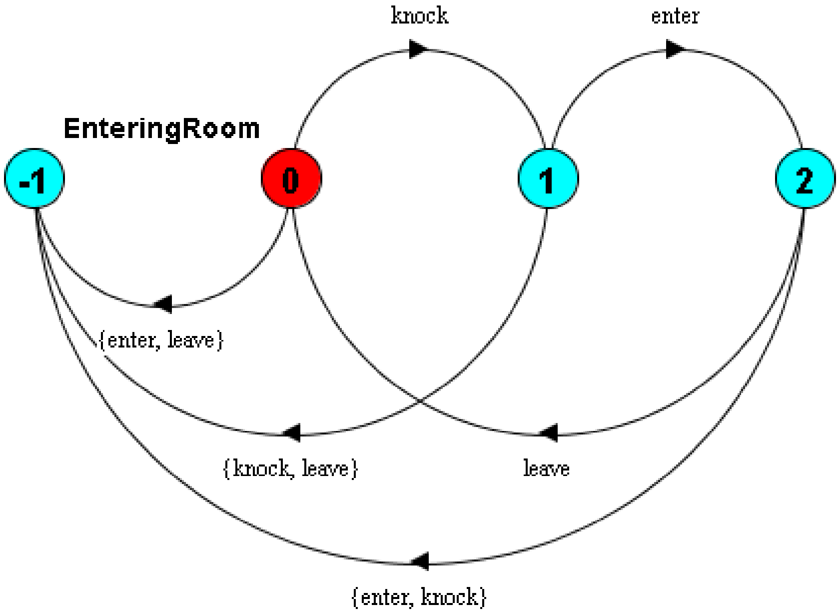
```
EnteringRoom=(knock->Enter
             |enter->ERROR),
Enter=(enter->leave->EnteringRoom
      |knock->ERROR).
```



Safety
Alerta especificado implicitamente

Um alternativa é se especificar (no processo) apenas o que se deseja realizar e prefixá-lo com a palavra-chave *property*. O compilador automaticamente gera uma transição para o estado -1 quando uma sequência inapropriada for executada.

```
property EnteringRoom=(knock->enter->leave->EnteringRoom).
```



*Escolhas não determinísticas não são suportadas.